

**SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
STROJNÍCKA FAKULTA**

**MOTO SHIELD: VÝVOJ PROGRAMÁTORSKÉHO
ROZHRANIA, IDENTIFIKÁCIA, RIADENIE A ÚPRAVA
DIDAKTICKÉHO PRÍSTROJA NA RIADENIE RÝCHLOSTI
MOTOROV**

Bakalárska práca

SjF-13432-92875

2020

Ján Boldocký

**SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
STROJNÍCKA FAKULTA**

**MOTO SHIELD: VÝVOJ PROGRAMÁTORSKÉHO
ROZHRANIA, IDENTIFIKÁCIA, RIADENIE A ÚPRAVA
DIDAKTICKÉHO PRÍSTROJA NA RIADENIE RÝCHLOSTI
MOTOROV**

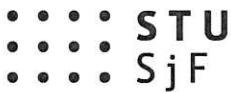
Bakalárska práca

SjF-13432-92875

Študijný program:	Automatizácia strojov a procesov
Študijný odbor:	kybernetika
Školiace pracovisko:	Ústav automatizácie, merania a aplikovanej informatiky
Vedúci záverečnej práce:	doc. Ing. Gergely Takács, PhD.
Konzultant:	Ing. Erik Mikuláš

Bratislava, 2020

Ján Boldocký



ZADANIE BAKALÁRSKEJ PRÁCE

Študent: **Ján Boldocký**
ID študenta: **92875**
Študijný program: **automatizácia a informatizácia strojov a procesov**
Študijný odbor: **kybernetika**
Vedúci práce: **doc. Ing. Gergely Takács, PhD.**
Konzultant: **Ing. Erik Mikuláš**
Miesto vypracovania: **ÚAMAI SjF STU v Bratislave**

Názov práce: **MotoShield: Vývoj programátorského rozhrania, identifikácia, riadenie a úprava didaktického prístroja na riadenie rýchlosťi motorov**

Jazyk, v ktorom sa práca vypracuje: slovenský jazyk

Špecifikácia zadania:

Úlohou študenta je vytvoriť programátorské rozhranie a inštruktážne príklady pre existujúci didaktický prístroj, tzv. "MotoShield". Jedná sa o miniaturizovaný modul pre mikroradičové prototypizačné dosky, ktorý obsahuje motor, prevodovku a inkrementálny snímač. Programátorské rozhranie je potrebné napísať v jazyku C/C++ pre prostredie Arduino IDE a taktiež pre prostredie MATLAB a Simulink. Študent vytvorí rôzne inštruktážne príklady na modelovanie, identifikáciu a spätnovázobné riadenie tohto systému. Práca musí obsahovať úpravu a návrh ďalšej možnej iterácie didaktického prístroja.

V rámci bakalárskej práce študent musí

- predstaviť zariadenie v relácii s existujúcou vedeckou literatúrou,
- otestovať funkčnosť prvej verzie hardvérového modulu,
- napísať programátorské rozhranie (application programming interface, API) pre ovládanie zariadenia v C/C++, MATLAB a Simulink,
- vytvoriť inštruktážne príklady vhodné na didaktické nasadenie (napr. identifikácia, spätnovázobné riadenie),
- využiť prostredie GitHub na manažovanie verzií softvéru,
- odporučiť možné hardvérové zmeny na vylepšenie prístroja.

Rozsah práce: **30-50 s.**

Riešenie zadania práce od: 17. 02. 2020
Dátum odovzdania práce: 19. 06. 2020

Ján Boldocký

študent



prof. Ing. Cyril Belavý, CSc.
vedúci pracoviska

Slovenská technická univerzita
v Bratislave
Dekanát Strojníckej fakulty
Útvar pedagogických činností
812 31 Bratislava, Nám. Slobody 17
-1-



prof. Ing. Cyril Belavý, CSc.
garant študijného programu

Čestné prehlásenie

Vyhlasujem, že som záverečnú prácu vypracoval samostatne s použitím uvedenej literatúry.

Bratislava, 18. júna 2020

Ján Boldocký

Ďakujem vedúcemu práce, doc. Ing. Gergely Takács, PhD., za odbornú pomoc, usmernenie a predovšetkým trpezlivosť pri vypracovaní bakalárskej práce.

Bratislava, 18. júna 2020

Ján Boldocký

Názov práce: MotoShield: Vývoj programátorského rozhrania, identifikácia, riadenie a úprava didaktického prístroja na riadenie rýchlosťi motorov

Kľúčové slová: jednosmerný motor, programátorské rozhranie, MotoShield, Arduino, PID regulátor, identifikácia

Abstrakt: V práci sa autor zaoberá analýzou a sprevádzkovaním vyhotoveného experimentálneho modulu MotoShield, určeného na didaktické účely v oblasti kybernetiky. Na začiatku sa čitateľ bližšie zoznámi s vyhotoveným hardvérom a s technickými údajmi jednotlivých prvkov modulu. Následne je popísaný postup vývoju programátorských rozhraní pre vývojové prostredia Arduino IDE, MATLAB a Simulink. Práca zahŕňa aj ukážkové príklady použitia vyhotovených programátorských rozhraní na ovládanie modulu a spätnoväzbové riadenie pomocou PID regulátora. Model systému bol získaný identifikáciou s analyticko-experimentálnym prístupom, pričom začatočný model systému bol odvodený na základe matematicko-fyzikálnych princípov. Na záver sú uvedené nedostatky hardvéru a navrhnutá je ďalšia iterácia hardvérovej zostavy modulu.

Title: MotoShield: Developing an Application Programming Interface, System Identification, Control and Modification of a Didactic Device for Motor Speed Control

Keywords: DC motor, application programming interface, MotoShield, Arduino, PID controller, system identification

Abstract: The thesis covers the analysis of the previously introduced experimental module MotoShield, originally designed for didactic purposes in cybernetics, as well as its practical usage. In the first part, the reader will familiarize himself with the hardware along with the technical specifications of the individual parts of the module. The development of the API for the Arduino IDE, MATLAB and Simulink development environments makes the majority of the thesis. The thesis describes the basic uses of the API as well as an example of the feedback control system featuring a PID controller. The mathematical model of the system was acquired by analytical-experimental system identification, whereby the initial model was derived. The final chapter of the thesis recounts the shortcomings of the hardware and proposes a design for the subsequent hardware iteration of the forthcoming module.

Obsah

Zoznam obrázkov	x
Zoznam skratiek	xi
Zoznam matematických symbolov	xii
Úvod	1
1 Východzí stav	3
1.1 Hardvér	3
1.2 Softvér	6
1.2.1 Prehľadnosť	6
1.2.2 Meranie výstupných veličín	7
2 Vývoj programátorského rozhrania	10
2.1 Arduino IDE API pre MotoShield	10
2.1.1 Metódy na ovládanie akčného člena	11
2.1.2 Metódy na meranie výstupných veličín	12
2.1.3 Pomocné metódy	16
2.2 MATLAB API pre MotoShield	17
2.3 Simulink API pre MotoShield	20
2.3.1 Ovládanie akčného člena	21
2.3.2 Meranie výstupných veličín	22
2.3.3 Blok systému MotoShield	23
2.3.4 Matematický model systému	27
3 Inštruktážne príklady	28
3.1 Odozva v otvorenej slučke	28
3.2 Spätnoväzbové riadenie a PID regulátor	30
3.3 Príklady riadenia pomocou PID regulátora	31
3.3.1 Arduino IDE	32
3.3.2 MATLAB	33
3.3.3 Simulink	36
4 Modelovanie a identifikácia	38
4.1 Tvorba modelu	38
4.1.1 Prenosová funkcia	39

4.1.2	Stavový priestor	39
4.2	Identifikácia	40
5	Nasledovná iterácia hardvéru modulu	44
6	Záver	47
Literatúra		48
A	Programátorské rozhrania - API	i
A.1	Arduino IDE API - Hlavičkový súbor	i
A.2	Arduino IDE API -Zdrojový súbor	ii
A.3	MATLAB API	iv
A.4	Zdrojový kód MATLAB funkcie pre blok MotoShield - Simulink API	vi
B	PID riadenie	vii
B.1	Arduino IDE	vii
B.2	MATLAB	viii
C	Identifikácia	ix
C.1	Stavový model - MATLAB	ix
C.2	Prenosová funkcia - MATLAB	x
C.3	Zber dát - Arduino IDE	xi

Zoznam obrázkov

1	Učebný prístroj s jednosmerným motorom od firmy Quanser (prebraté z [17])	2
1.1	Arduino UNO pripojené s MotoShieldom (prebraté z [12])	3
1.2	Výstupný signál z inkrementálneho snímača	4
1.3	Zariadenie L293D s logickou štruktúrou	4
1.4	Schéma elektrického obvodu pomôcky MotoShield (prebraté z [12])	5
1.5	MotoShield (prebraté z [12])	6
2.1	Ilustrácia zhrnutia obslúh externého a cyklického prerušenia	15
2.2	Bloky knižnice MotoShield	20
2.3	Nastavenie smeru otáčania – súčasť bloku Actuator Write	21
2.4	Zápis akčného zásahu – súčasť bloku Actuator Write	22
2.5	Meranie uhlovej rýchlosťi – Sensor Read	23
2.6	Blok systému s externou referenčnou hodnotou --MotoShield	24
2.7	Maska a Mask Editor bloku systému – MotoShield	25
2.8	Blok prenosovej funkcie – MotoShield TF	27
3.1	Odozva systému MotoShield na skokovú zmenu	29
3.2	Schéma inštrukčného príkladu InputsOutputs v prostredí Simulink	30
3.3	Výsledok PID riadenia v prostredí Arduino IDE	34
3.4	Výsledok PID riadenia v prostredí MATLAB	36
3.5	Bloková schéma PID riadenia v prostredí Simulink	37
3.6	Výsledok PID riadenia v prostredí Simulink	37
4.1	Amplitúdová a frekvenčná charakteristika hornopriepustného filtra	41
4.2	Výsledok filtrovania signálu odberu elektrického prúdu	42
4.3	Porovnanie identifikovanej prenosovej funkcie s nameranými hodnotami	43
4.4	Porovnanie identifikovaného stavového modelu s nameranými hodnotami	43
5.1	Vyčítaný signál odberu prúdu z 10-bitového analógovo-digitálneho prevodníka	45
5.2	Schéma elektrického obvodu ďalšej iterácie modulu	46
5.3	Tlačená obvodová doska - PCB	46

Zoznam skratiek

AD	Analógovo-digitálny (prevodník)
API	Programátorské rozhranie (angl. Application Programming Interface)
CMOS	(angl. Complementary Metal Oxide Semiconductor)
IIR	Nekonečná impulzová odozva (angl. Infinite Impulse Response)
ISR	Obsluha prerušenia (angl. Interrupt Service Routine)
MIMO	Viacero-vstupov viacero-výstupov (angl. Multiple-input Multiple-output)
PCB	Tlačená obvodová doska (angl. Printed circuit board))
PID	Proporcionálne integračne derivačný (regulátor) (angl. Proportional–integral–derivative)
PPR	Počet pulzov na otáčku (angl. Pulses Per Revolution)
PSE	Kladná nábehová hrana (signálu) (angl. Positive Signal Edge)
PWM	Šírkovo modulovaný (signál) (angl. Pulse-width Modulation)
RPM	Otáčky za minútu (angl. Revolutions per minute)
SI	Medzinárodná sústava jednotiek (franc. Système international)
SISO	Jednotlivý-vstup jednotlivý-výstup (angl. Single-input Single-output)
SMT	Technológia povrchovej montáže (angl. Surface-mount technology)
THT	Technológia osadzovanej montáže (angl. Through-hole technology)
TTL	(angl. Transistor-transistor Logic)
UART	Univerzálny asynchronný sériový prenos (angl. Universal Asynchronous Receiver-transmitter)

Zoznam matematických symbolov

Symbol	Veličina	Jednotka (použitá v práci)
A	Zosilnenie	-
b	Koeficient viskózneho tlmenia	Nms
e	Regulačná odchýlka	-
$G(s)$	Prenosová funkcia	-
$I(s)$	Elektrický prúd (obrazová oblast)	A
i	Elektrický prúd	A
I_s	Elektrický prúd	A
J	Moment zotrvačnosti	kg·m ²
K_d	Derivačná konštanta	-
K_e	Koeficient elektromotorickej sily	Vrad ⁻¹ s ⁻¹
K_i	Integračná konštanta	-
K_p	Proporcionálna konštanta	-
K_t	Koeficient momentu motora	NmA ⁻¹
k	Index vzorky	-
k_{\min}	Počet vzoriek v jednej minúte	-
kT	Diskrétny čas	s
L	Indukčnosť	H
M	Krútiaci moment	Nm
M_h	Hnací moment	Nm
M_t	Trecí moment	Nm
N	Množina prirodzených čísel	-
n	Počet impulzov	-

R	Elektrický odpor	Ω
r	Referenčná hodnota	-
s	Komplexná premenná	-
t	Spojity čas	s
T_d	Derivačná časová konštanta	-
T_i	Integračná časová konštanta	-
T_{\min}	Čas jednej minúty	s
t_0	Doba aktívneho stavu	-
t_p	Periódna PWM signálu	-
T_s	Periódna vzorky	s
u	Akčný zásah	-
U_{RMS}	Efektívna hodnota napäťia	V
U_z	Vstupné napätie	V
V	Amplitúda PWM signálu	V
V_{OUT}	Výstupné napätie senzoru prúdu	V
x	Stavová veličina	-
y	Výstupná veličina	-
α_0	Uhol jednej otáčky	rad
\mathcal{E}	Elektromotorická sila	V
θ	Uhol otočenia	rad
$\dot{\theta}$	Uhlová rýchlosť	$\text{rad}\cdot\text{s}^{-1}$
$\ddot{\theta}$	Uhlové zrýchlenie	$\text{rad}\cdot\text{s}^{-2}$
τ	Diferenciálne malý časový úsek	s

Úvod

V oblasti technických vied sú praktické cvičenia dôležitou súčasťou procesu výučby. Re realizujú sa za pomoci učebných prístrojov, vďaka ktorým sa študenti bližšie zoznámia s problematikou danej témy a jasne pochopia ciele a postupy metód pri riešení úloh. Učebné prístroje robia proces výučby atraktívnejším a taktiež aj podnecujú motiváciu u študentov. Konkrétnie pre výučbu automatizačnej techniky je veľmi dôležité mať skúsenosti s fyzickými systémami, keďže veľakrát na úspešné riadenie nepostačuje analytický prístup, ale je potrebná intuitívna schopnosť.

Množstvo firiem sa zaobráva výrobou a predajom komerčných učebných prístrojov na výuku automatizačnej techniky, ktoré sú výborne skoncipované, kvalitne vyrobené a dôkladne zdokumentované. Avšak, ich nevýhodou je, že sú cenovo neprístupné pre mnohé verejné školy. Na druhú stranu, vyskytuje sa veľké množstvo tzv. „urob si sám“ projektov na internete, ktoré väčšinou neposkytujú žiadnu dokumentáciu ani softvérové rozhranie.

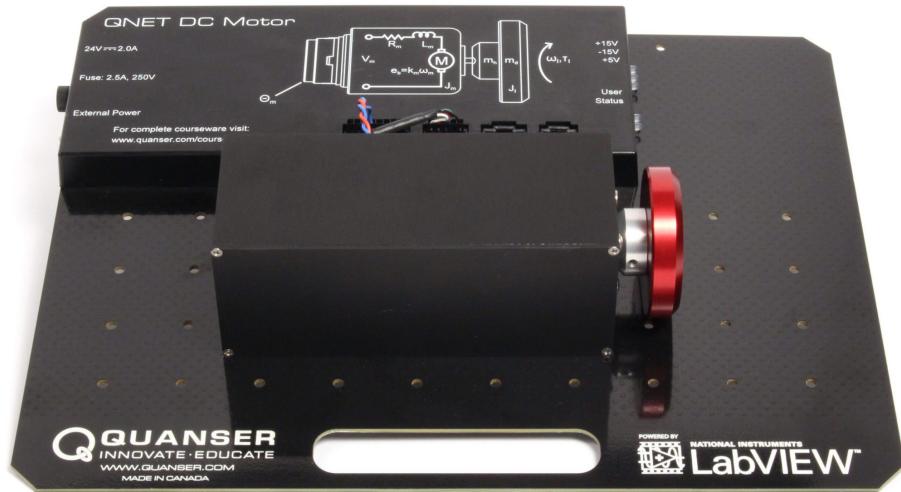
Projekt AutomationShield je alternatívou. Ponúka široký výber lacných didaktických prístrojov na výuku automatizačnej techniky. Prístroje sú vyhotovené vo forme kompaktných externých rozšírení pre mikroradičové dosky rodiny Arduino, presnejšie pre dosky Uno, Mega, Leonardo, Due a Zero. Platforma Arduino a projekt AutomationShield zdieľajú rovnakú ideológiu otvoreného zdroja (open-source). Kompletná dokumentácia a softvérové rozhranie sú pre každý prístroj projektu AutomationShield voľne dostupné na platforme GitHub [9]. Pre väčšinu prístrojov je dostupné softvérové rozhranie v prostrediacich Arduino IDE, MATLAB a Simulink.

Cieľom tejto práce je napísanie softvérové rozhranie s inštruktážnymi príkladmi pre existujúci prístroj projektu AutomationShield, pomenovaný MotoShield. Jedná sa o modul, ktorého účelom je riadenie uhlovej rýchlosťi integrovaného jednosmerného motora, pričom je uhlová rýchlosť meraná pomocou inkrementálneho snímača a odber elektrického prúdu pomocou meracieho odporu. *Prístroj MotoShield bol navrhnutý a rozpracovaný v práci Tibora Konkolyho [12], pričom konečný stav prístroja z predchádzajúcej práce je predstavený v Kap. 1.*

MotoShield patrí medzi najjednoduchšie prístroje projektu AutomationShield z hľadiska regulovania výstupnej veličiny. Predovšetkým z dôvodu nízkej zotrvačnosti stavu systému a preto je odporúčaný medzi prvými na vyskúšanie v procese učenia kybernetiky. Pripojený MotoShield na dosku Arduino UNO je znázornený na Obr. 1.1.

Výhoda MotoShield-u v porovnaní s inými dostupnými učebnými prístrojmi je predovšetkým kompaktnosť. Rozmery obvodovej dosky modulu sú totožné s rozmermi dosky Arduino UNO (68.6 mm × 53.3 mm) [1]. Na rozdiel od väčšiny prístrojov, MotoShield je jednodielny, čo je veľkou výhodou najmä pri použití v školských učebniach. Jediný porovnatelný produkt s MotoShield-om je prístroj *QNET DC Motor 2.0* od firmy *Quanser*,

ktorého nedostatkom je podpora softvérového rozhrania iba pre prostredie LabVIEW a taktiež aj vysoká cena. Tento produkt už nie je komerčne dostupný.

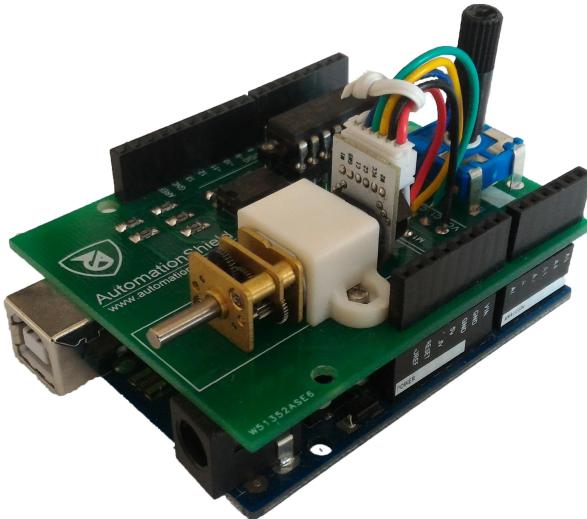


Obr. 1: Učebný prístroj s jednosmerným motorom od firmy Quanser (prebraté z [17])

1 Východzí stav

1.1 Hardvér

Pomôcka MotoShield je modul navrhnutý v predchádzajúcej práci [12] pre mikrokontrolerové prototypizačné dosky rodiny Arduino so štandardom logických úrovní 5 V TTL¹, čo znamená že logická jednotka je interpretovaná napäťovou úrovňou 5 V a logická nula úrovňou 0 V. Z hľadiska rozmerovej kompatibility je modul vyhotovený pre dosky so standardizovaným rozmiestnením vstupno-výstupných pinov aké má napr. doska Arduino UNO.

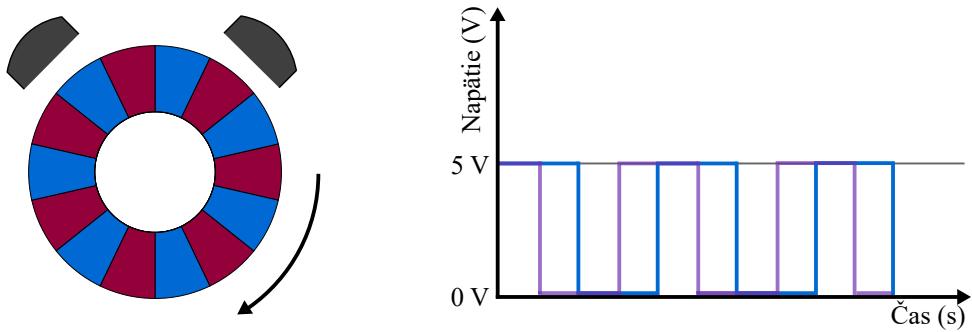


Obr. 1.1: Arduino UNO pripojené s MotoShieldom (prebraté z [12])

Akčným členom je jednosmerný motorček s komutátorom od výrobcu DFRobot. Pracovné napätie motora je 6 V, pri ktorom v nezaťaženom stave maximálny odber prúdu motora je 60 mA. Pri takomto príkone by mal motor dosiahnuť maximálnu uhlovú rýchlosť 15000 ot/min [8]. Keďže motor zahŕňa aj prevodovku s prevodovým pomerom 380:1, na prednom hriadele rýchlosť otáčania bude 380-krát menšia ako rýchlosť motora.

Zo zadnej strany motora sa nachádza sedem-pólové magnetické koliesko a rotačný inkrementálny snímač. Snímač pracuje na princípe Hallovho javu a obsahuje dve Hallove sondy, ktoré v prítomnosti severného magnetického pólu prepustia elektrické napätie. Výsledkom otáčania sa magnetického kolieska je výstupný signál z kanálov inkrementálneho snímača s obdlžnikovým charakterom Obr. 1.2.

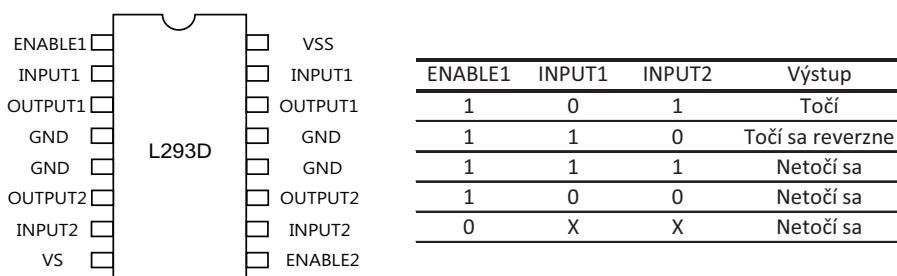
¹Transistor-transistor logic



Obr. 1.2: Výstupný signál z inkrementálneho snímača

Motorová jednotka obsahuje šesť terminálov. Prvý M1 a posledný M2 sú svorky motora, druhý VCC a štvrtý GND sú na napájanie inkrementálneho snímača. Tretí vývod C1 a štvrtý C2 sú výstupné kanály snímača [12], ktoré sú potom pripojené k mikrokontroléru na digitálne vstupy D3 a D4.

Na ovládanie smeru otáčania motora je použitý H mostík L293D [19]. Jedná sa o dvojkanálový H mostík v DIP16 prevedení, schopný viesť prúdy do 600 mA pri napäti 4.5 V až 36 V. Tento komponent zahŕňa diódy proti spätným prúdom a napäťovým špičkám pri zapínaní motora a taktiež aj ochranu proti prehriatiu pre teploty nad 150 °C. Pre MotoShield je použitý iba jeden kanál, čiže osem vývodov. Vývod VSS je na napájanie vnútornej logiky zariadenia L293D, kým na vývod VS sa privádzajú prúdy na napájanie motora z 5 V výstupu mikroradiča. Samotný motor je pripojený na vývody OUTPUT1 a OUTPUT2. Vnútorné tranzistory na nastavenie polarity prúdu sú pripojené na vývody INPUT1 a INPUT2 a ovládané sú cez digitálne výstupy mikroradiča D6 a D7. Regulácia rýchlosť je realizovaná cez vývod ENABLE1, ktorý má primárnu úlohu v logickej štruktúre zariadenia, totož motor sa točí iba v prípade keď sa na vývode ENABLE1 nachádza logická jednotka v podobe napäťa Obr. 1.3 [12]. Vhodnou metódou ovládania rýchlosť v tomto prípade je šírkovo modulovaný signál (angl. Pulse-width modulation, PWM), privádzaný na vývod ENABLE1 vysielaný z digitálneho výstupu mikroradiča D5.

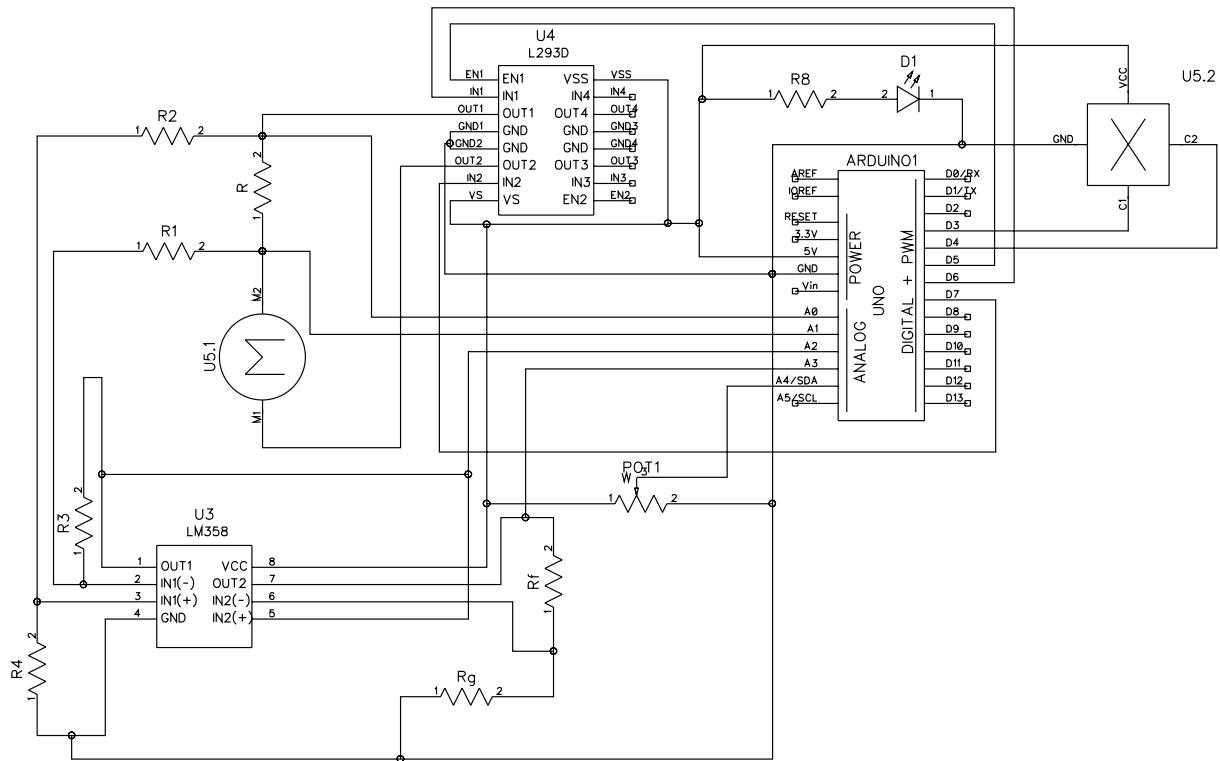


Obr. 1.3: Zariadenie L293D s logickou štruktúrou

Ďalšia periféria elektrického obvodu je určená na nepriame meranie odberu prúdu motora. Táto pozostáva z dvojkanálového operačného zosilňovača LM358 v PDIP8 prevedení a meracieho odporu (angl. shunt resistor) s hodnotou 10Ω , ktorý je zapojený do série s motorom. Zmeraním úbytku napäťa cez merací rezistor je možné odhadnúť spotrebu

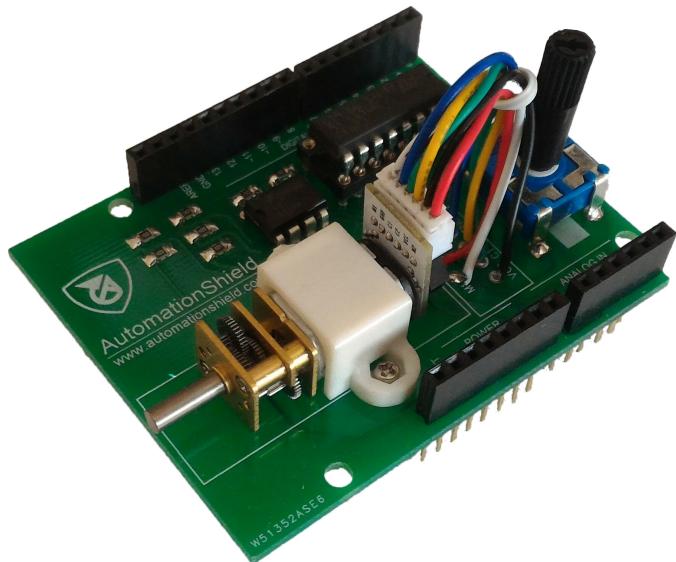
elektrického prúdu, keďže platí, že prúd je priamoúmerný úbytku napäťia a nepriamoúmerný odporu, ktorý ho zapríčinil. Prvý operačný zosilňovač je v prevedení diferenčného zosilňovača, ktorého účelom je na výstupe získať rozdiel vstupných napäťí. Keďže vstupné signály sú napäťia pred a po meracom odpore, výstupom bude hodnota úbytku napäťia. Signál úbytku napäťia je potom privedený na vstupný terminál druhého zosilňovača, ktorý tento signál zosilní o hodnotu určenú Rov. (1.1) [12]. Zo schémy elektrického obvodu Obr. 1.4 vidno usporiadanie konfiguračných odporov R_1 až R_4 pre diferenčný a R_f , R_g pre neinvertujúci zosilňovač. Výstupné signály z diferenčného a neinvertujúceho zosilňovača sú mikroradiču privádzané na analógovo-digitálny prevodník cez piny A2 a A3.

$$A = 1 + \frac{R_f}{R_g} = 1 + \frac{10 \Omega}{5.1 \Omega} = 2.96 \quad (1.1)$$



Obr. 1.4: Schéma elektrického obvodu pomocný MotoShield (prebraté z [12])

Posledná súčiastka obvodu je potenciometer s odporovou vrstvou $10\text{ k}\Omega$, ktorého účelom je manuálne nastavenie referenčnej hodnoty. Napätie z bežca potenciometra je mikroradiču privádzané cez analógový pin A4 Obr. 1.4.



Obr. 1.5: MotoShield (prebraté z [12])

1.2 Softvér

V predchádzajúcej práci autora Konkolyho [12] bola napísaná knižnica programového rozhrania (angl. Application Programming Interface, API) pre vývojové prostredie Arduino IDE. Toto vývojové prostredie je predovšetkým určené na programovanie dosiek rodiny Arduino použitím programovacích jazykov C a C++. Knižnica bola vytvorená podľa štandardného štýlu písania [2], zložená z dvoch súborov. Pri objektovo orientovanom programovaní v jazyku C++ by *hlavičkový súbor* mal byť prvoradý v procese čítania knižnice, keďže dáva jej jasný prehľad. Pod tým chápeme prehľad tried, štruktúr, modifikátorov prístupu, metód, preprocesorových príkazov, premenných atď. Hlavičkový súbor sa identifikuje rozšírením názvu súboru (angl. filename extension) „*.h“.

1.2.1 Prehľadnosť

Analýzou hlavičkového súboru knižnice „MotoShield.h“ sa najprv stretнемe s preprocesorovým príkazom `#define`, ktorý slúži na vytvorenie tzv. makier.

```
#define priklad 1
```

Zjednodušene, uvedený príkaz sa vykoná pred samotnou kompliaciou programu a jeho výsledkom bude nahradenie textu `priklad` číslom 1 v kóde. Autor predchádzajúcej knižnice vhodne použil tento príkaz na vytvorenie symbolických konštánt pre pomenovanie pinov. Týmto spôsobom by autor mal pomenovať všetky konštanty v kóde, pretože sa takto udrží prehľadnosť kódu a zároveň sa nealokuje pamäť zbytočným počtom premenných.

Ďalším dôvodom nadmerného počtu deklarovaných premenných je bezdôvodné priradenie hodnôt pri definovaní metód v *zdrojovom súbore*, ktorého rozšírenie názvu je „*.cpp“. Tento nedostatok sa vyskytuje aj pri metóde `referenceRead`.

```

float MotoClass::referenceRead(){
    _referenceRead = analogRead(MOTO_RPIN);
    referenceValue =
        → AutomationShield.mapFloat(_referenceRead,0.00,1023.00,0.00,100.00);
    return referenceValue;
}

```

Lepším spôsobom zápisu uvedenej metódy je vložiť funkciu `analogRead` do prvého vstupného argumentu metódy `mapFloat` a toto deklarovať ako návratovú hodnotu. Týmto zápisom by sa získal rovnaký výsledok s výhodou vyšej účinnosti a prehľadnosti kódu. Taktiež sa v knižnici prejavuje aj hromadenie neopodstatnených metód `motorON` a `motorOFF`, ktorých účel je roztočenie motora maximálnym príkonom a jeho vypnutie. Dané metódy je možné vylúčiť z knižnice, keďže sú zastupiteľné `actuatorWrite` metódou, popísanou na strane 12.

Z dôvodu jednoznačnosti, na písanie programových rozhraní má projekt AutomationShield jasne stanovené pravidlá, medzi ktoré patrí aj názvoslovie metód. Na čítanie výstupných veličín sa používa vzor `sensorRead*`, na vyslanie vstupných `actuatorWrite*` a napriek tomu sa autor predchádzajúceho rozhrania vo viacerých prípadoch týmto neriadil.

1.2.2 Meranie výstupných veličín

Metódy, pomocou ktorých meriame stavové veličiny systému patria k najdôležitejším pri návrhu programového rozhrania. Keďže metóda na meranie odberu prúdu `readCurrent` vychádza zo súčinu konštanty a hodnoty úbytku napäťia, metóda ktorou získame úbytok napäťia `readVoltage` bude rozhodujúca pre presnosť odhadu hodnoty prúdu. Hoci bola navrhnutá periféria hardvéru na spracovanie signálu úbytku napäťia, autor knižnice ju v metóde `readCurrent` nepoužil. Získanie hodnoty úbytku napäťia realizoval odčítaním napäti pred a po meracom rezistore.

```

float MotoClass::readVoltage(){
    ADCR1 = analogRead(MOTO_U1);
    ADCR2 = analogRead(MOTO_U2);
    if(ADCR1 > ADCR2){
        ADCU = ADCR1 - ADCR2;
    }
    if(ADCR2 > ADCR1){
        ADCU = ADCR2 - ADCR1;
    }
    k = (5.00 / 1023.00);
    V = ADCU * k;
    return V;
}

```

Na meranie uhlrovej rýchlosťi motora v API „MotoShield.h“ boli napísané dve metódy. Obe používajú počítadlo `counter`, ktoré je inkrementované v rámci obsluhy prerušenia (angl. Interrupt Service Routine, ISR) `countTicks`. Obsluha prerušenia je nastavená vstavanou funkciou Arduino knižnice `attachInterrupt(MOTO_C1, countTicks, FALLING)` v rámci inicializačnej metódy `begin`. Keďže je pin inkrementálneho snímača nastavený obrátene príkazom `pinMode(MOTO_C1, INPUT_PULLUP)` a režim spúšťania

obsluhy je FALLING [3], obsluha prerušenia bude spúšťaná pri nábehu kladnej hrany signálu² na pine MOTO_C1.

```
void MotoClass::countTicks(){
    Bstate = digitalRead(MOTO_C2);
    counter ++ ;
}
```

Premenná `Bstate`, určená na detegovanie smeru otáčania, nadobúda hodnotu logickej úrovne pinu druhého výstupu inkrementálneho snimača a v API nebola použitá.

Prvá z metód na meranie uhlovej rýchlosťi motora je `durationTime`, z ktorej vystupuje čas trvania jednej otáčky predného hriadeľa v milisekundách.

```
float MotoClass::durationTime(){
    rev = 380;
    count = counter;
    cValue = count - previousCount;
    t = millis();
    if(cValue >= rev){
        revTime = t;
        durTime = revTime - prevTime;
        previousCount = count;
    }
    prevTime = revTime;
    return durTime * 7; // (7 * 380 = 2660)
}
```

Ked'že sa v tomto programátorskom rozhraní impulzy Hallovej sondy zapisujú kumulatívne počas behu programu, na meranie rýchlosťi bolo potrebné odčítať súčasnú hodnotu počítadla od predchádzajúcej. Aktuálna hodnota počítadla je uložená v premennej `cValue`, ktorá je následne v rámci podmienky testovaná, či nadobúda hodnotu väčšiu alebo rovnú ako 380. Keď sa táto podmienka splní, pomocou funkcie `millis` sa zistí kumulatívny čas a podobným spôsobom sa odčíta časový interval, za ktorý bolo zaznamenaných 380 impulzov. Návratová hodnota bude 7-násobok odčítaného intervalu, lebo jedna otáčka predného hriadeľa vydá 2660 impulzov. Dôvod, prečo je v podmienke číslo 380 je neznámy a návratová hodnota by bola presnejšia, keby v podmienke bolo skutočné číslo počtu impulzov pri jednej otáčke. Metóda `durationTime` si ľahko nájde uplatnenie a účel, preto je považovaná za nadbytočnú.

Druhá z metód na meranie uhlovej rýchlosťi je `readRevolutions`, ktorej návratová hodnota je počet otáčok predného hriadeľa za minútu. Správny výpočet meranej veličiny je možný iba v prípade, že je metóda zavolaná iba raz za vzorku, pričom do vstupného argumentu metódy musí byť uvedený vzorkovací čas v milisekundách.

```
float MotoClass::readRevolutions(int Time){
    Count = counter;
    noInterrupts();
    rValue = Count - prevC;
    h = 1000 / Time;
    constant = ((float(h) * 60.00) / 2660.00);
    REV = float(rValue) * constant;
    prevC = Count;
```

²angl. positive signal edge

```
    interrupts();
    return REV;
}
```

Zakázanie externého prerusenia funkciou `noInterrupts` v predchádzajúcej práci [12] nebolo zdôvodnené a nepovažujem ho za povinné pre uvedenú metódu. Metóda musí byť volaná vo vzorkovacom algoritme a nutné je uvedenie vzorkovacej periódy, preto túto metódu považujem za neflexibilnú a nevhodnú na použitie.

2 Vývoj programátorského rozhrania

V predchádzajúcej práci [12] pre modul MotoShield bolo napísané nepostačujúce programátorské rozhranie pre prostredie Arduino IDE, preto jedným z cieľov tejto práce bude napísať nové. Keďže v rámci projektu AutomationShield by každý modul mal obsahovať aj programátorské rozhranie pre prostredia MATLAB a Simulink, druhá a tretia podkapitola bude venovaná týmto softvérovým nástrojom.

2.1 Arduino IDE API pre MotoShield

Pri tvorbe programátorského rozhrania pre prostredie Arduino IDE je vhodné napísat knižnicu, ktorá zhrnie všetky potrebné metódy. Štruktúra knižnice je štandardná [2], inými slovami, pozostáva z hlavičkového „*.h“ a zdrojového súboru „*.cpp“.

V hlavičkovom súbore je prvým krokom zstrojiť tzv. podmienené zahrnutie¹ preprocessorovými príkazmi. Účelom podmieneného zahrnutia je vyhýbanie sa chybám komplátora pri viacnásobnom zahrnutí hlavičky do programu.

```
// prvá časť
#ifndef id
#define id
// podmienená časť
#endif
// druhá časť
```

V uvedenom kóde sa prvá a druhá časť bezpodmienečne skompilujú, kým podmienená časť sa skompiluje iba v prípade, že identifikátor `id` neboli definované v prvej časti. Pri prvom zahrnutí sa identifikátor zdefinuje, a tak sa zabezpečí kompliacia podmienenej časti iba jedenkrát.

V podmienenej časti sa pomocou príkazu preprocesora `#include <Arduino.h>` zahrnie knižnica, ktorá disponuje vstavanými Arduino funkciami ako je napr. funkcia `digitalWrite` a podobne. Ďalším uplatnením preprocessorových direktív je vytvorenie tzv. symbolických konštánt [16], ktoré umožnia prehľadnosť kódu bez zbytočnej alokácie pamäte. Takýmto spôsobom sú definované všetky konštanty v knižnici, pričom napr. `#define SHUNT 10.0` definuje hodnotu meracieho odporu.

Pre moduly projektu AutomationShield je programátorské prostredie písané objektovo orientovanou formou. Pod tým chápeme, že funkcie a premenné sú súčasťou triedy, ktorej názov zvyčajne býva „MenoShieldClass“, kde „Meno“ je názov konkrétneho Shieldu. Trieda je programátorom definovaný dátový typ pomocou ktorého sa tvoria inštancie

¹angl. conditional inclusion

alebo tzv. objekty [22]. Tieto môžu disponovať členmi triedy, ktorých prístup je deklarovaný ako `public`. Syntax definovania triedy je nasledovná:

```
class MotoShieldClass{  
public:  
// verejný prístup  
private:  
// prístup v rámci triedy  
};
```

Metódy a premenné určené na volanie mimo triedy sú deklarované identifikátorom prístupu `public` a tie, ktoré sú volané iba v rámci triedy, sú deklarované identifikátorom prístupu `private`.

2.1.1 Metódy na ovládanie akčného člena

Jedna z najjednoduchších metód v tomto programovom rozhraní je `setDirection`. Jej účelom je nastaviť smer otáčania motora. Deklarovaná `void setDirection(bool direction = true)` je v rámci triedy za verejným identifikátorom prístupu. Na základe dátového typu `void` je zrejmé, že metóda nemá návratovú hodnotu. Metóda čaká jeden vstupný argument dátového typu `bool`, ktorého predvolená hodnota je `true` v prípade, že pri volaní argument nie je upresnený. V zdrojovom súbore rozhrania je metóda definovaná nasledovne:

```
void MotoShieldClass::setDirection(bool direction = true) {  
    if (direction) {           //--pravotočivý  
        digitalWrite(MOTO_DIR_PIN1, 0);  
        digitalWrite(MOTO_DIR_PIN2, 1);  
    }  
    else {                    //--ľavotočivý  
        digitalWrite(MOTO_DIR_PIN1, 1);  
        digitalWrite(MOTO_DIR_PIN2, 0);  
    }  
}
```

Kedže je definovaná mimo triedy, potrebné je špecifikovať rozsah, do ktorého patrí. Toto je realizované operátorom rozlíšenia rozsahu „::“. Telom metódy je jednoduchá podmienka na testovanie hodnoty vstupného argumentu. Pri kladnej hodnote vstupného argumentu sa motoru udelí pravotočivý smer otáčania a obrátene. Pomocou metódy `digitalWrite` sa pinom pripojeným k L293D pridelí logická úroveň v podobe napäťia. Logickú štruktúru komponentu L293D môžete vidieť na Obr 1.3.

Metóda na určenie pohonu akčného člena je pomenovaná `actuatorWrite` podľa pravidiel nomenklatúry projektu AutomationShield [10]. Z deklarácie metódy `void actuatorWrite(float percentValue)` je zrejmé, že metóda nemá návratovú hodnotu a nutne očakáva jeden vstupný argument dátového typu `float`. Uvedená hodnota do vstupného argumentu je očakávaná v percentánoch.

```
void MotoShieldClass::actuatorWrite(float percentValue){  
    if(percentValue < minDuty && percentValue != 0)  
        percentValue = minDuty; // Ak je vstup 0, motor je vypnutý  
    analogWrite(MOTO_UPIN,  
               AutomationShield.percToPwm(percentValue));  
}
```

Analýzou tela metódy v zdrojovom súbore je možno vidieť, že pomocou jednoduchej podmienky je korigovaná saturácia akčného člena. Podmienka porovnáva hodnotu vstupného argumentu `percentValue` s minimálnou hodnotou akčného zásahu, pri ktorej sa motor začne otáčať. Minimálna hodnota akčného zásahu je interpretovaná premennou `minDuty`, ktorá vychádza z kalibračnej metódy² a v prípade, že kalibrácia nebola uskutočnená premenná bude nadobúdať nulovú hodnotu. Ak je hodnota vstupného argumentu menšia ako minimálna, na akčný člen bude poslaný minimálny akčný zásah. Druhá časť podmienky hovorí o tom, že na jej splnenie hodnota vstupného argumentu nesmie byť nulová. Týmto sa zabezpečí vypnutie motora v prípade že sa metóda zavolá s nulovým vstupným argumentom. Na koniec, prívod PWM signálu na akčný člen je uskutočnený funkciou `analogWrite`, ktorej prvý vstupný argument je číslo digitálneho pinu a druhý je hodnota činiteľa plnenia v 8-bitovom rozlíšení. Prevod hodnoty z percentuálnej na 8-bitovú je realizovaný metódou `AutomationShieldClass::percToPwm` [9].

Ďalšia metóda na ovládanie akčného zásahu je pomenovaná `actuatorWriteVolt` z dôvodu, že za vstupný argument očakáva efektívnu hodnotu napäťa šírkovo modulovaného signálu. Výpočet efektívnej hodnoty napäťa je určený pomocou Rov. (2.1), kde U_{RMS} je efektívna hodnota napäťa (angl. root mean square voltage), V je konštantná hodnota amplitúdy PWM signálu, t_p je perióda a $t_o(t)$ je doba aktívneho stavu [20]

$$U_{\text{RMS}}(t) = V \sqrt{\frac{t_o(t)}{t_p}} \implies t_o(t) = \frac{U_{\text{RMS}}^2(t)}{V^2} t_p. \quad (2.1)$$

Metódy `actuatorWrite` a `actuatorWriteVolt` sú takmer rovnaké, jediný rozdiel je v hodnote minimálneho akčného zásahu, ktorá je v metóde `actuatorWriteVolt` interpretovaná premennou `minVolt` a prevodom hodnoty vstupného argumentu na hodnotu 8-bitého rozlíšenia.

```
void MotoShieldClass::actuatorWriteVolt(float voltageValue){
    if(voltageValue < minVolt != 0) voltageValue = minVolt;
    analogWrite(MOTO_UPIN,sq(voltageValue)*255.0/sq(REF));
}
```

2.1.2 Metódy na meranie výstupných veličín

V tejto časti budú popísané metódy na meranie úbytku napäťa resp. odberu prúdu a uhlovej rýchlosťi motora.

Meranie odberu prúdu

Metódy na meranie úbytku napäťa boli napísané tri z dôvodu využitia všetkých schopností hardvéru. Všetky tri majú verejný prístup, návratovú hodnotu dátového typu `float` a neočakávajú vstupné argumenty. Prvá je pomenovaná `sensorReadVoltage`, táto vráti absolútnu hodnotu rozdielu napäťa pred a po meracom odpore. Druhá sa volá `sensorReadVoltageAmp1` a vráti hodnotu výstupu diferenčného zosilňovača, čiže úbytok napäťa. Tretia metóda vráti najpresnejšiu hodnotu úbytku napäťa a pomenovaná je `sensorReadVoltageAmp2`. Jej návratová hodnota vychádza z výstupného signálu

²Kalibračná metóda je rozpísaná na strane 16

neinvertujúceho zosilňovača, ktorého hodnota zosilnenia je 2.96 (-), Rov. (1.1). Hodnota zosilnenia je kvôli prehľadnosti definovaná symbolickou konštantou `AMP_GAIN`. Definícia metódy v zdrojovom súbore je pomerne jednoduchá. Pomocou funkcie `analogRead` sa vyčíta hodnota z príslušného pinu analógovo-digitálneho prevodníka s 10-bitovým rozlíšením, ktorá sa potom prevedie na volty koeficientom $\frac{5}{1023}$ V. Na získanie reálnej hodnoty úbytku napäťa je potrebné ešte tento signál vydeliť hodnotou zosilnenia `AMP_GAIN`.

```
float MotoShieldClass::sensorReadVoltageAmp2() {
    return analogRead(MOTO_YAMP2)*5.0/1023.0/AMP_GAIN;
}
```

Získanie hodnoty odberu prúdu je založené iba na numerickej úprave návratovej hodnoty funkcie `sensorReadVoltageAmp2`, čo zahrňa metóda `sensorReadCurrent`. Z úbytku napäťa sa prúd získa vydelením s hodnotou meracieho odporu a navyše je výstupná hodnota násobená číslom 1000, čo reprezentuje prevod jednotky z A na mA.

```
float MotoShieldClass::sensorReadCurrent() {
    return MotoShield.sensorReadVoltageAmp2()/SHUNT*1000.0;
}
```

Meranie uhlovej rýchlosťi

Spôsob merania uhlovej rýchlosťi je založený na externom a cyklickom prerušení. Externé prerušenie znamená spúštanie obsluhy prerušenia³ zmenou napäťovej úrovne na príslušnom vstupnom pine, čo je vhodné pri použití inkrementálneho snímača ktorý ma výstupný signál obľžnikového charakteru (Obr. 1.2). Obsluha externého prerušenia je v rozhraní deklarovaná ako súkromná metóda, pomenovaná `_InterruptServiceRoutine`. Z definície vidíme, že táto metóda predstavuje iba jednu inkrementovaciu inštrukciu premennej `count` a nemá žiadnu návratovú hodnotu.

```
void MotoShieldClass::_InterruptServiceRoutine(){
    count++;
}
```

Na definovanie obsluhy externého prerušenia je potrebné použiť funkciu s názvom `attachInterrupt`, ktorá je súčasťou knižnice „Arduino.h“ integrovanej do vývojového prostredia Arduino IDE. Prvý argument funkcie umožňuje voľbu čísla pinu, pričom pre dosku Arduino UNO sú na externé prerušenie vhodné digitálne piny D2 a D3. Druhý vstupný argument funkcie očakáva uvedenie mena funkcie, ktorá predstavuje obsluhu prerušenia. Tretím sa zadefinuje režim spúšťania, teda podmienka, kedy sa obsluha prerušenia bude spúštať. Použitý je režim `CHANGE`, pri ktorom sa obsluha prerušenia spustí aj pri kladnej, aj zápornej nábehovej hrane signálu. Na rozdiel od režimov `FALLING` a `RISING`, pri ktorých sa obsluha spustí buď iba na zápornú, alebo iba na kladnú nábehovú hranu, výhodou použitého režimu je dvakrát väčšia hustota impulzov na otáčku. Metóda použitá ako obsluha cyklického prerušenia je pomenovaná `_InterruptSample`. Táto bude spúštaná raz za každú vzorkovaciu periódu. Na definovanie cyklického prerušenia je použité rozhranie „Sampling.h“, ktoré je súčasťou projektu AutomationShield [9]. Pomocou tohto rozhrania prehľadne a jednoducho určíme vzorkovaciu periódu a obsluhu prerušenia. Vzorkovacia perióda je určená zavolaním metódy `SamplingClass::period` s

³angl. Interrupt Service Routine

uvedením periódy v mikrosekundách ako vstupný argument. Obsluha prerusenia je zadefinovaná metódou `SamplingClass :: interrupt`, ktorá ako vstupný argument očakáva funkciu.

Inštrukcie zahrnuté v obsluhe cyklického prerusenia sú tri. Najprv sa hodnota počítadla `count` priradí pomocnej pamäťovej premennej `counted` a tak sa zaznamená počet impulzov predchádzajúcej vzorky. Potom sa hodnota počítadla vynuluje a na konci sa priradí kladná hodnota pomocnej premennej `stepEnable`. Pomocná premenná je uplatnená v príkladoch riadenia, Kap. 3.3.

Metódy, ktoré predstavujú obsluhu cyklického a externého prerusenia sú deklarované ako statické pomocou kľúčového slova `static`. Dôvodom je chyba pri kompliacii, ktorá nastala kvôli prehláseniu funkcie triedy za tzv. „callback“ funkciu. Keď funkciu triedy deklarujeme za statickú, funkcia už nebude potrebovať inštanciu na volanie ale bude patriť triede, čiže sa bude správať ako globálna pre danú triedu.

```
private:  
    static void _InterruptServiceRoutine();  
    static void _InterruptSample();
```

Keďže statické metódy môžu manipulovať iba statickými premennými, všetky premenné použité v obsluhe cyklického resp. externého prerusenia musia byť takto deklarované. Okrem toho, statické premenné by mali byť aj inicializované mimo triedy. Ale keďže od verzie jazyka C++17 je možné vykonať inicializáciu statických premenných spolu s deklaráciou pomocou slova `inline`, inicializácia mimo triedy je zbytočná [7]. Hodnoty premenných sa v rámci prerusenia nečakane menia, a preto tieto premenné musia byť označené kvalifikátorom `volatile`. Toto zabezpečí volanie premennej z operačnej pamäte, keďže je architektúra mikropočítača 8-bitová, mikropočítač nedokáže v jednom kroku vyčítať viac ako jeden byte. Preto sú premenné počítadla deklarované ako 8-bitové kladné celé čísla [4].

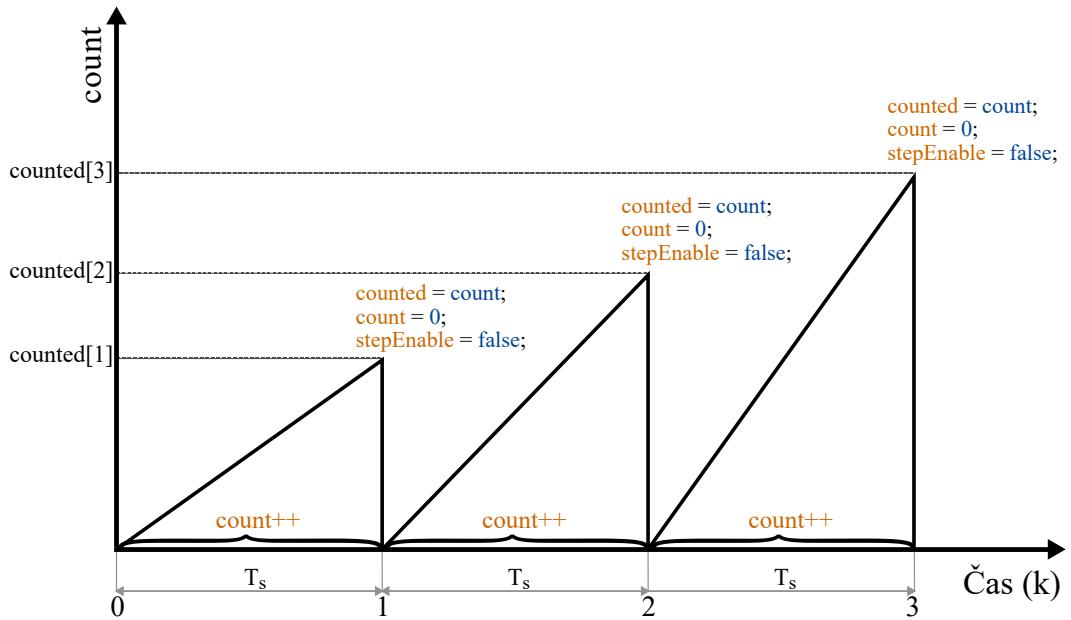
```
static inline volatile uint8_t count;  
static inline volatile uint8_t counted;  
static inline volatile bool stepEnable;
```

Funkcie na definovanie obslúh externého a cyklického prerusenia sú zavolané v inicializačnej metóde `begin` triedy `MotoShieldClass`.

Definovanie obslúh prerusenia môže byť realizované aj pomocou tzv. lámbd alebo anonymných „callback“ funkcií. Lambdy sú definované priamo v argumente, pričom v hranatých zátvorkách sú uvedené premenné, ktoré sa prenesú do vnútra lambdy. V jednoduchých zátvorkách je uvedený vstupný argument a v množinových sa nachádzajú samotné príkazy funkcie. V prípade použitia lámbd na definovanie obslúh prerusenia metódy `_InterruptSample` a `_InterruptServiceRoutine` by stratili význam. Definovanie obslúh pomocou lámbd by sa realizoval nasledovne:

```
Sampling.interrupt([count](void){counted = count; count = 0; stepEnable = true;});  
attachInterrupt(digitalPinToInterruption(MOTO_YPIN1), [](void){count++;}, CHANGE);
```

Pomocou ďalších dvoch metód, `sensorReadRPM` a `sensorReadRPMPerc`, je hodnota uhlovej rýchlosťi upravená tak, aby sa na výstupe získali jednotky používané v praxi. Návratovou hodnotou metódy `sensorReadRPM` sú otáčky na minútu (angl. Revolutions per Minute, RPM). Prevod jednotky uhlovej rýchlosťi z počtu impulzov za periódou na



Obr. 2.1: Ilustrácia zhrnutia obslúh externého a cyklického prerušenia

počet otáčok za minútu je znázornený v Rov. (2.2), kde n je počet impulzov, T_s je períoda vzorky a k_{min} je počet vzoriek v jednej minúte. Číslo 14 reprezentuje počet impulzov jednej otáčky, T_{min} čas jednej minúty a α_o uhol jednej otáčky:

$$\frac{\alpha_o}{T_{min}} = k_{min} \frac{n}{T_s \cdot 14} \quad | \quad k_{min} = \frac{T_{min}}{T_s}. \quad (2.2)$$

V programovom prevedení výpočet vyzerá rovnako, s výnimkou že počet impulzov za sekundu je vyjadrený premennou `counted`, ktorá je pretypovaná⁴ na dátový typ `float` kvôli jednoznačnosti dátových typov vo výpočte.

```
float MotoShieldClass::sensorReadRPM(){
    return (float)counted/14.0*_K;
}
```

Metóda `sensorReadRPMPerc` vráti uhlovú rýchlosť otáčania v percentách. Na preškálovanie do percent potrebujeme poznáť aspoň dva body závislosti medzi stupnicami. V tomto prípade to bude hodnota maximálnej, čiže 100 % uhlovej rýchlosťi, uloženej v premennej `maxRPM` a hodnota minimálnej, čiže 0 % uhlovej rýchlosťi, uloženej v premennej `minRPM`. Preškálovanie je riešené pomocou metódy `AutomationShieldClass ::mapFloat`, ktorá vráti hodnotu v percentách. Na obmedzenie návratových hodnôt je použitá aj metóda `AutomationShieldClass ::constrainFloat` [9]. Hodnoty premenív `maxRPM` a `minRPM` sú získané kalibračnou metódou `MotoShieldClass ::calibration`.

```
float MotoShieldClass::sensorReadRPMPerc(){
    return AutomationShield.constrainFloat(AutomationShield.mapFloat(counted,
        (float)minRPM, (float)maxRPM, 0.0, 100.0), 0.0, 100.0);
}
```

⁴Pretypovanie je lokálna zmena dátového typu pre nasledujúcu inštrukciu. Realizovaná je uvedením dátového typu pred premenou v zátvorkovej konvencii.

2.1.3 Pomocné metódy

Metóda, ktorá by sa pri použití prístroja MotoShield mala zavolať prvá je inicializačná metóda `begin`. Jej účelom je vykonať inštrukcie nastavenia, pričom sa najprv nastaví režim pinov príkazom `pinMode`, potom sa metódou `MotoShieldClass::setDirection` zvolí pravotočivý smer otáčania a tým pádom metóda voľby smeru otáčania nebude povinná pri aplikácii. Ostatné príkazy sú vysvetlené v časti Meranie uhlovej rýchlosťi. Pri volaní tejto metódy je možné špecifikovať periódnu cyklického prerušenia do vstupného argumentu v milisekundách a v prípade že pri volaní argument zostane prázdný, perióda nadobudne predvolenú hodnotu 50 milisekúnd.

```
void MotoShieldClass::begin(float _Ts = 50.0){ //Milisekundy
    pinMode(MOTO_UPIN, OUTPUT);
    pinMode(MOTO_DIR_PIN1,OUTPUT);
    pinMode(MOTO_DIR_PIN2,OUTPUT);
    pinMode(MOTO_YPIN1, INPUT);
    pinMode(MOTO_YPIN2, INPUT);
    pinMode(MOTO_RPIN, INPUT);
    pinMode(MOTO_YV1, INPUT);
    pinMode(MOTO_YV2, INPUT);
    pinMode(MOTO_YAMP1, INPUT);
    pinMode(MOTO_YAMP2, INPUT);
    setDirection(true);
    Sampling.period(_Ts*1000.0);           // Prevedenie na mikrosekundy
    _K=60000.0/_Ts;                      // 60 000 - minuta v milisekundach
    Sampling.interrupt(_InterruptSample);
    attachInterrupt(digitalPinToInterrupt(MOTO_YPIN1), _InterruptServiceRoutine,
    → CHANGE);
}
```

V prípade, že užívateľ potrebuje merať uhlovú rýchlosť v percentách, musí povinne najprv vykonať kalibráciu, čiže zavolať metódu `calibration`. Týmto sa získa maximálna a minimálna hodnota uhlovej rýchlosťi otáčania a minimálny akčný zásah. Postup inštrukcií metódy je popísaný komentárimi vedľa príkazov:

```
void MotoShieldClass::calibration(){
    actuatorWrite(100);      // Maximálny akčný zásah
    delay(1000);            // Oneskorenie na roztočenie motora
    maxRPM = counted;       // Meranie max. uhlovej rýchlosťi
    actuatorWrite(0);        // Vypnutie motora
    delay(500);             // Oneskorenie na zastavenie motora
    int i = 15;              // Akčný zásah predpokladaný za min. v percentách
    do{                     // Nekonečný cyklus
        actuatorWrite(i);   // Zápis akčného zásahu
        delay(300);          // Oneskorenie na roztočenie
        if(counted >= 4){   // Podmienka pohybu
            delay(1000);     // Oneskorenie na roztočenie
            minDuty = i;      // Záznam min. akčného zásahu
            minRPM = counted; // Záznam min. rýchlosťi
            minVolt = AREF * sqrt(minDuty/100.0);
            actuatorWrite(0);  // Vypnutie motora
            break;            // Zastavenie cyklu
        }
        i++;                 // Inkrementovanie predpokladaného minima
}
```

```

}while(1);
}

```

Hodnoty oneskorenia vyhovujú iba pre kalibráciu pri nezaťaženom stave motora. V prípade zaťaženia by čas oneskorení musel byť predĺžený. V ďalšej iterácii programátorského rozhrania by metóda mala obsahovať vstupný argument, ktorý by predstavoval moment zotrvačnosti záťaže, pomocou ktorého sa čas oneskorení adekvátne predĺži. Posledná metóda, `referenceRead`, je univerzálna pre každý prístroj projektu AutomationShield. Deklarovaná je ako verejná s návratovou hodnotou dátového typu `float`. Jej účelom je vrátiť hodnotu polohy natočenia bežca potenciometra v percentách. Používa sa pri manuálnom nastavení referencie.

2.2 MATLAB API pre MotoShield

V tejto časti bude popísaný postup tvorby programátorského rozhrania pre prostredie MATLAB a významné odlišnosti v logickej štruktúre metód od metód programátorského prostredia napísaného pre Arduino IDE. Na programovanie prototypizačných dosiek rodiny Arduino pomocou prostredia MATLAB existuje balík *MATLAB Support Package for Arduino Hardware*. Tento zahŕňa základné funkcie a ďalšie knižnice, použité v tomto programátorskom rozhraní [15].

V prostredí MATLAB tiež existujú abstraktné celky, tzv. triedy, ku ktorým patria premenné, metódy, objekty atď. Inými slovami, skriptovací jazyk MATLAB je taktiež objektovo orientovaný. Definovanie triedy sa syntaktický realizuje nasledovne: `classdef MenoTriedy < handle obsah triedy end`. Takto vytvorená trieda bude vlastne podriedou supertryedy `handle`, pričom získava prístup k metódam supertryedy. V obsahu triedy `MotoShield` je najprv definované jej vlastníctvo pomocou funkcie `properties`, pričom ako argument sú uvádzané vlastnosti definovaných prvkov. Prvá skupinka vlastníctva zahŕňa definovanie verejných objektov a premenných a v druhej skupine vlastníctva sú definované konštenty triedy.

```

classdef MotoShield < handle
    properties(Access = public)
        arduino;
        encoder;
        minDuty=25;
        minRPM;
        maxRPM;
    end
    properties(Constant)
        MOTO_YPIN1 = 'D2';
        MOTO_YPIN2 = 'D3';
        MOTO_UPIN = 'D5';
        MOTO_DIR_PIN1 = 'D6';
        MOTO_DIR_PIN2 = 'D7';
        MOTO_YV1 = 'A0';
        MOTO_YV2 = 'A1';
        MOTO_YAMP1 = 'A2';
        MOTO_YAMP2 = 'A3';
        MOTO_RPIN = 'A4';
        RES = 10.0;
    end

```

```

AMP_GAIN = 2.96;
PPR = 7;
end

```

Definovanie metód je umožnené funkciou `methods`. V programátorskom rozhraní pre prístroj MotoShield sú všetky metódy verejné, čo je upresnené atribútom `Access = public`. Rovnako ako aj pri funkcií `properties`, obsah funkcie je ukončený príkazom `end`. V obsahu funkcie `methods` sa nachádzajú definované metódy. Sú to vlastne *funkcie*, ktoré si za prvý vstupný argument vyžadujú objekt danej triedy. Výnimka platí pre metódy s atribútom `Static`, ktoré pri volaní nepotrebuju objekt.

Prvá definovaná metóda v programátorskom rozhraní je inicializačná metóda `begin`. Z definície vidíme, že okrem objektu triedy `MotoShieldObject`, metóda očakáva dva ďalšie vstupné argumenty `Port` a `Board`, ktoré sú potom odovzdané funkcií `arduino` pri tvorbe objektu pre používanú prototypizačnú dosku. Tieto argumenty spresňujú typ Arduino dosky a číslo komunikačného (COM) portu, cez ktorý je pripojená. Funkcia `arduino`, obsahuje aj vstupné argumenty `*, 'Libraries', 'rotaryEncoder'`) na zahrnutie knižnice „`rotaryEncoder`“, ktorá rozšíri možnosti mikroradiča, presnejšie umožní manipuláciu s inkrementálnym snímačom. Takto vytvorený objekt je potom priradený k premennej `arduino`, ktorá pri volaní potrebuje uvedenie objektu s predponou `MotoShieldObject` a bodkovou konvenciou, pretože je nestatický člen triedy. Nastavenie režimu jednotlivých pinov je vykonané príkazom `configurePin`, ktorý vyžaduje uvedenie objektu mikroradiča ako prvý vstupný argument, potom číslo pinu a na koniec uvedenie zvoleného režimu. Zoznam pinov je definovaný pomocou konštánt triedy, a preto je pri volaní potrebné uviesť objekt triedy. Zoznam jednotlivých režimov pre piny je dostupný v MathWorks dokumentácii [13]. Po nakonfigurovaní pinov sa vytvorí objekt knižnice „`rotaryEncoder`“ príkazom `rotaryEncoder`, ktorý vyžaduje štyri vstupné argumenty. Prvý je objekt použitého mikroradiča, druhý a tretí sú piny kanálov a štvrtý je počet impulzov na otáčku (angl. Pulses per Revolution, PPR) resp. počet kladných pôlov magnetického kolieska. V prípade MotoShield-u magnetické koliesko má sedem kladných pôlov. Na koniec, podobne ako pri API pre Arduino IDE je zavolaná funkcia `setDirection(MotoShieldObject)` na nastavenie predvoleného smeru otáčania.

```

function begin(MotoShieldObject, Port, Board)
MotoShieldObject.arduino = arduino(Port,Board,'Libraries','rotaryEncoder');
configurePin(MotoShieldObject.arduino,MotoShieldObject.MOTO_YPIN1,'Interrupt');
...
...
...
configurePin(MotoShieldObject.arduino,MotoShieldObject.MOTO_DIR_PIN2,'DigitalOutput');
MotoShieldObject.encoder=rotaryEncoder(MotoShieldObject.arduino,MotoShieldObject.MOTO_」
→ _YPIN1,MotoShieldObject.MOTO_YPIN2,MotoShieldObject.PPR);
setDirection(MotoShieldObject);
end

```

Dôvodom zahrnutia knižnice „`rotaryEncoder`“ sú metódy na meranie uhlovej rýchlosťi. Vytvorený objekt `encoder` je schopný zavolať tri funkcie, z ktorých funkcia `readSpeed` vráti uhlovú rýchlosť v jednotke otáčky za minútu. Táto funkcia je samostatne postačujúca na meranie uhlovej rýchlosťi, avšak kvôli jednotnosti názvoslovia projektu AutomationShield [10] je zabalená do metódy `sensorReadRPM`. Pomocou identifikátora `rpm`

je návratová hodnota funkcie `readSpeed` deklarovaná za návratovú hodnotu funkcie `sensorReadRPM`.

```
function rpm = sensorReadRPM(MotoShieldObject)
    rpm = readSpeed(MotoShieldObject.encoder);
end
```

Podobne vyzerá aj metóda `sensorReadRPMperc`, z ktorej vystupuje uhlová rýchlosť vyjadrená v percentách. Rovnako ako aj pri API pre Arduino IDE, na preškálovanie sú potrebné hodnoty maxima a minima uhlovej rýchlosťi, ktoré sa získavajú pomocou kalibračnej metódy `calibration(MotoShieldObject)`. Na preškálovanie je použitá funkcia `map`, ktorej výstupná hodnota je obmedzená na maximálnu a minimálnu hodnotu funkciou `constrain`. Funkcie použité v metóde `sensorReadRPMperc` sú súčasťou projektu AutomationShield, ktorých autorom je Peter Chmurčiak.

```
function rpmPerc = sensorReadRPMperc(MotoShieldObject)
    rpmPerc = constrain(map(readSpeed(MotoShieldObject.encoder), MotoShieldObject.minRPM, MotoShieldObject.maxRPM, 0, 100), 0, 100);
end
```

Kalibračná metóda `calibration` má niekoľko rozdielností v porovnaní s kalibračnou metódou v prostredí Arduino IDE. Pred cyklom hľadania najmenšej hodnoty akčného zásahu, čiže najnižšej možnej uhlovej rýchlosťi sa vynuluje počítadlo impulzov enkódera vykonaním príkazu `resetCount(MotoShieldObject.encoder)`. Lokálnej premennej `i` sa priradí hodnota predpokladaného minima akčného zásahu 0.15, lebo funkcia na vyslanie PWM signálu `writePWMDutyCycle` očakáva vstupnú hodnotu 0 až 1. V cykle sa potom postupne zapisuje čoraz silnejší akčný zásah a testuje sa podmienka na detegovanie pohybu. Táto porovnáva hodnotu počítadla impulzov, získanú príkazom `readCount(MotoShieldObject.encoder)`, ktorej absolútна hodnota je porovnávaná s číslom 8. Porovnávaná je absolútна hodnota, lebo pri pravotočivom pohybe hodnota počítadla dekrementuje a tak nadobudne zápornú hodnotu. Číslo 8 v tomto prípade reprezentuje dve sedminy otáčky, keďže knižnica „`rotaryEncoder`“ používa oba kanály inkrementálneho snímača s režimom `CHANGE`.

Prvá verzia didaktického prístroja MotoShield má kanály inkrementálneho snímača privedené na nevhodnú dvojicu pinov D3 a D4. Vhodné piny na externé prerušenie pre dosku Arduino UNO sú D2 a D3. Knižnica „`rotaryEncoder`“ používa oba piny a preto je nutné drôtikom prepojiť pin D4 s pinom D2.

```
function calibration(MotoShieldObject)
    writePWMDutyCycle(MotoShieldObject.arduino, MotoShieldObject.MOTO_UPIN, 1);
    pause(1); % oneskorenie na roztočenie motora max. rýchlosťou
    MotoShieldObject.maxRPM=readSpeed(MotoShieldObject.encoder);
    writePWMDutyCycle(MotoShieldObject.arduino, MotoShieldObject.MOTO_UPIN, 0);% vypnutie
    pause(0.5); % oneskorenie na zastavenie motora
    i = 0.15; % predpokladaný min akčný zásah - 15%
    resetCount(MotoShieldObject.encoder); % vynulovanie počítadla
    while(1) % nekonečný cyklus
        writePWMDutyCycle(MotoShieldObject.arduino, MotoShieldObject.MOTO_UPIN, i);
        pause(0.3); % oneskorenie na min. roztočenie
        if abs(readCount(MotoShieldObject.encoder)) > 8 % detegovanie pohybu
            pause(1); % oneskorenie na roztočenie min. zásahom
    end
end
```

```

MotoShieldObject.minDuty = i*100; % vyčítanie min. akčného zásahu
MotoShieldObject.minRPM = readSpeed(MotoShieldObject.encoder); % min. RPM
actuatorWrite(MotoShieldObject,0); % vypnutie motora
break; % ukončenie cyklu
end
i=i+0.01;
end      % koniec cyklu
end        % koniec metódy

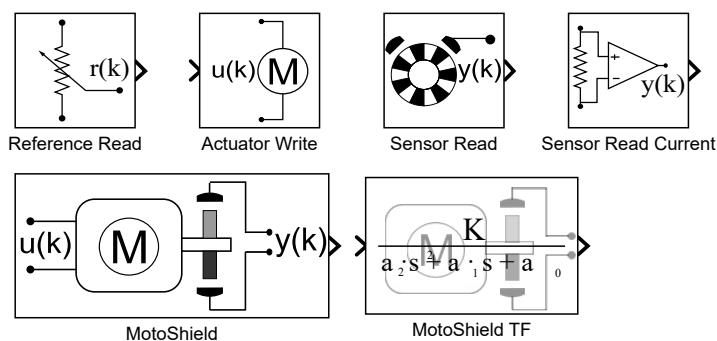
```

2.3 Simulink API pre MotoShield

Vývojové prostredie Simulink je založené na grafickom rozhraní blokových schém s prispôsobiteľnou sadou blokových knižníc [14]. Programovanie dosiek rodiny Arduino prostredníctvom Simulink-u je umožnené balíkom *Simulink Support Package for Arduino Hardware*. V rámci projektu AutomationShield bola vytvorená knižnica, ktorá obsahuje dedikované bloky pre niektoré moduly projektu. Sady blokov pre príslušné moduly môžu obsahovať bloky s nasledovnými funkciami:

- meranie výstupných veličín – Sensor Read,
- ovládanie akčného zásahu – Actuator Write,
- čítanie referenčnej hodnoty potenciometra – Reference Read,
- blok systému s názvom rovnakým ako je názov modulu, ktorý zhrnie zápis akčného zásahu a meranie regulovanej (výstupnej) veličiny,
- matematický model systému.

Pri tvorbe jednotlivých blokov je potrebné vytvoriť aj tzv. masku bloku. Maska je grafické okno, ktoré zobrazuje popis funkčnosti a interaktívne prvky pomocou ktorých je možné meniť nastavenia resp. vnútornú štruktúru bloku. Taktiež umožňuje aj pridanie vlastného ilustračného obrázku ktorým bude daný blok označený. Tvorba a použitie masky je bližšie popísané v sekcií Blok systému na str. 24.



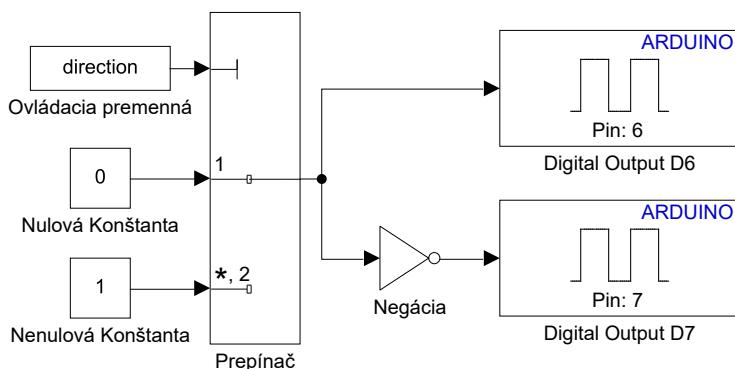
Obr. 2.2: Bloky knižnice MotoShield

Bloky **Reference Read** a **Sensor Read Current** sú v rámci programátorského rozhrania najjednoduchšie. Vnútorná schéma pozostáva z bloku **Analog Input**, ktorého

výstup predstavuje hodnotu napäťia na príslušnom pine analógovo digitálneho prevodníka v 10-bitovom rozlíšení. Výstupný signál je potom adekvátnym násobkom prevedený na inú veličinu.

2.3.1 Ovládanie akčného člena

Pod ovládaním akčného člena chápeme nastavenie smeru otáčania a určenie činiteľa plnenia PWM signálu. Na stanovenie smeru otáčania je potrebné nastaviť logické úrovne výstupných digitálnych pinov D6 a D7 podľa tabuľky⁵ na Obr. 1.3. Nastavenie výstupnej hodnoty digitálneho pinu je realizované blokom **Digital Output**, ktorý čaká vstupný signál buď nulový alebo nenulový. Číslo pinu je špecifikované do textového poľa v maske bloku **Digital Output**. Časť schémy na nastavenie smeru otáčania zahrňa ešte aj bloky konštánt s nulovou, nenulovou a premennou hodnotou **direciton**, viacportový prepínač a logickú negáciu. Viacportový prepínač prepúšťa jeden zo signálov z ľavej strany na pravú, v závislosti od hodnoty ovládacieho vstupu. Ovládací vstup je prvý zhora na, ktorý je privádzaná hodnota premennej **direciton**. Premenná nadobúda hodnoty 1 alebo 2, čo je stanovené voľbou parametra masky.

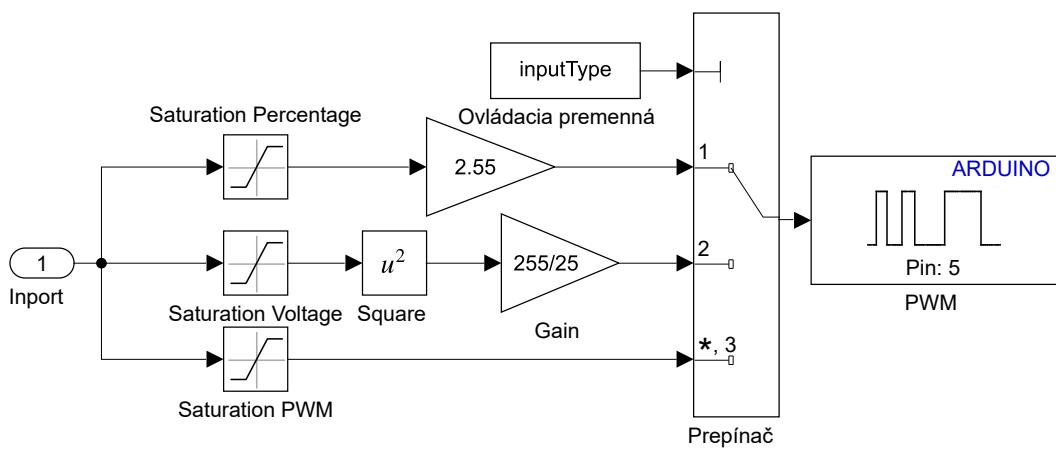


Obr. 2.3: Nastavenie smeru otáčania – súčasť bloku Actuator Write

Druhá časť bloku **Actuator Write** na zápis PWM signálu používa blok **PWM**, ktorého číslo pinu sa špecifikuje rovnako ako pri bloku **Digital Output**. Blok **PWM** očakáva jeden vstupný signál s hodnotou nie väčšou ako 255 a zároveň nezápornou. Toto číslo predstavuje činiteľ plnenia⁶ šírkovo modulovaného signálu. Hodnota signálu vo vnútri bloku pochádza zo vstupu do samotného bloku **Actuator Write**. Signál sa do vnútra bloku vnáša prvkom **Import**, ktorý je k troj-vstupovému prepínaču privedený v troch rôznych vetvách. Pri každej vetve sa najprv blokom **Saturation** obmedzí hodnota signálu príslušným intervalom. Potom sa signál upraví tak, aby do prepínaču vchádzal v 8-bitovom rozlíšení, čiže 0 až 255. Rovnako, ako aj pri nastavovaní smeru, hodnota ovládacej premennej **inputType** vychádza z masky bloku **Actuator Write**. Pri zvolení vstupnej hodnoty v percentách je aktívna prvá vetva, druhá vetva je v prípade napäťia a tretia v prípade hodnoty 8-bitového rozlíšenia. Prepočet zo strednej hodnoty napäťia na 8-bitovú stupnicu vyplýva zo vzťahu v Rov. (2.1).

⁵INPUT1 je pripojený na pin D6 a INPUT2 na D7

⁶angl. duty cycle



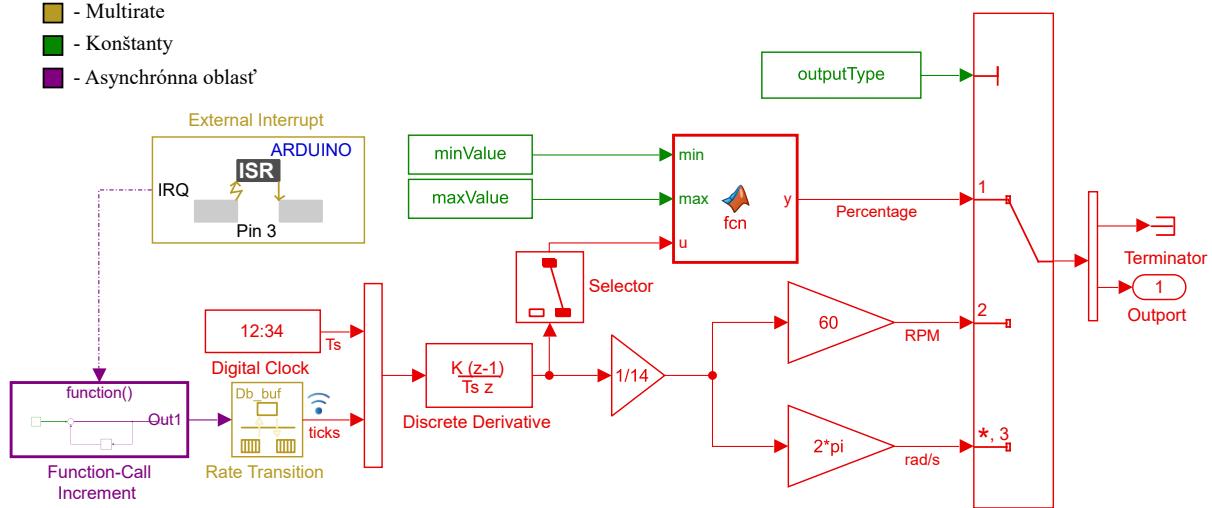
Obr. 2.4: Zápis akčného zásahu – súčasť bloku Actuator Write

2.3.2 Meranie výstupných veličín

Na meranie uhlovej rýchlosťi motora je použitý inkrementálny snímač pri ktorom je vhodné použiť počítadlo inkrementované v rámci obsluhy externého prerušenia. Knižnica podpory Arduino hardvéru pre Simulink zahŕňa blok **External Interrupt**, ktorý deteguje externé prerušenie na určenom digitálnom pine. Číslo pinu, na ktorom sa deteguje prerušenie spolu s režimom spúšťania sa špecifikuje v maske bloku. Výstupný signál z bloku prerušenia tzv. „function-call event“ je generovaný v okamihoch prerušenia. Je to vlastne špeciálny typ signálu, schopný spúšťať blok „funciton-call subsystem“. Vo vnútornej štruktúre bloku **Sensor Read** je spúšťaný subsystém pomenovaný **Function - Call Increment**. Tento v sebe zahŕňa jednoduchú inkrementovaciu schému.

Výstupný signál počítadla je z hľadiska času neurčitý, a preto ho je potrebné zo-synchronizovať. To je realizované kombináciou **Digital Clock** a **Rate Transition** blokov. Z bloku **Digital Clock**, vystupuje hodnota aktuálneho času simulácie vo vzorkovanom formáte, pričom periódou vzorky sa uvádzajú do textového poľa masky. Blok **Rate Transition** umožňuje prenos dát medzi blokmi v rôznych časových oblastiach. Kombinovaný signál je z hľadiska času určitý a predstavuje štrnásťnásobok počtu vykonaných otáčok. Deriváciou sa získava uhlová rýchlosť, ale keďže je signál vysielaný v diskrétnom čase, na deriváciu je použitý **Discrete Derivative** blok. Signál uhlovej rýchlosťi je ďalej odvádzaný na prepínač v troch vetvách. Pri prvej vetve sa signál privedza do bloku MATLAB funkcie, kde sa preškáluje na percentá pomocou funkcií **map** a **constrain**. Kalibračné hodnoty, uložené v premenných **minValue** a **maxValue**, musia byť definované programátorom v maske. Druhá a tretia vetva majú spoločného deliteľa, po ktorom signál bude predstavovať uhlovú rýchlosť v otáčkach za sekundu. Na získanie otáčok za minútu sa signál potom vynásobí číslom 60. V tretej vetve sa násobkom 2π získava uhlová rýchlosť v jednotkách radián za sekundu. Vo vnútornej štruktúre bloku **Sensor Read** na Obr. 2.7 sú farbami znázornené signály a bloky rôznych časových oblastí, označené legendou v ľavom hornom rohu. Vetva výstupného signálu je zvolená v závislosti od premennej **outputType**, ktorej hodnota závisí od zvolenej možnosti v maske. Do vonkajšieho prostredia je signál prenesený prvkom **Outport**.

- - Diskretná oblasť s periódou T_s
- - Multirate
- - Konštanty
- - Asynchronná oblasť



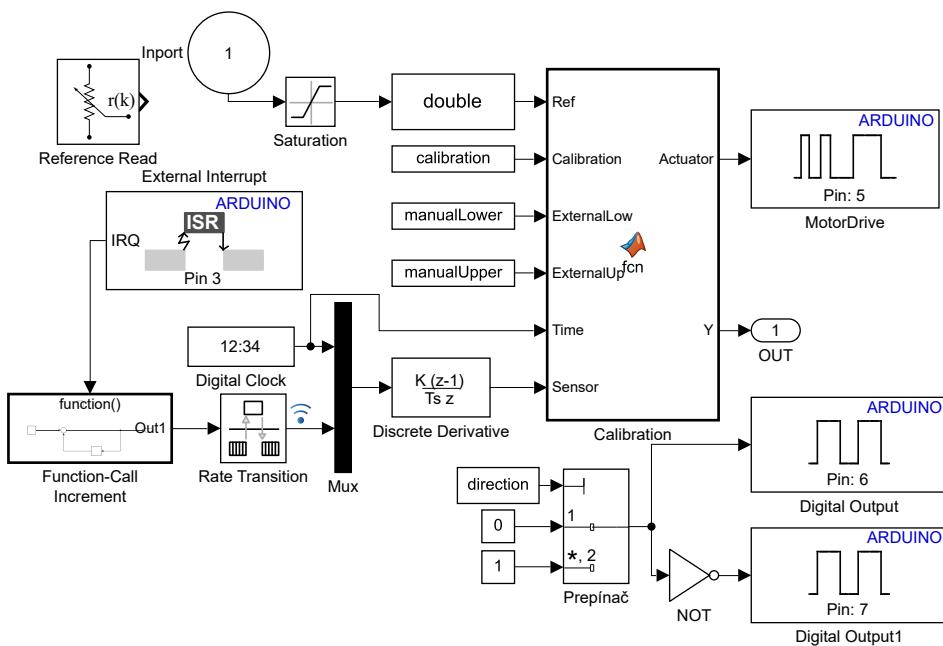
Obr. 2.5: Meranie uhlovej rýchlosť – Sensor Read

2.3.3 Blok systému MotoShield

Blok pomenovaný **MotoShield** reprezentuje zhrnutie blokov na ovládanie akčného člena a meranie uhlovej rýchlosťi. Hodnota výstupného a vstupného signálu je vyjadrená v percentách. Blok je charakteristický tým, že pri spustení dokáže vykonať kalibráciu na získanie kalibračných hodnôt. Referenčná hodnota je nastaviteľná buď ako externý signál vstupujúci do bloku, alebo manuálnym nastavením bežca potenciometra.

Vnútorná štruktúra bloku **MotoShield** obsahuje jednotlivé časti blokov **Sensor Read** a **Actuator Write**, pričom kľúčový prvok štruktúry je blok MATLAB funkcie **Calibration**, ktorý berie šesť vstupných signálov. Medzi vstupné signály patria informácie o kalibrácii, signál referenčnej hodnoty, časový signál a signál nameranej uhlovej rýchlosťi. Do kalibračných informácií patria minimálna a maximálna uhlová rýchlosť otáčania a príznak, ktorý hovorí o tom, či sa kalibrácia uskutoční alebo kalibračné hodnoty zadá programátor. Výstupné signály z bloku **Calibration** sú dva, kde prvý nesie signál akčného zásahu a je privedený k bloku **PWM**, ktorý druhý je upravený signál uhlovej rýchlosťi. Signál uhlovej rýchlosťi je prenesený do vonkajška pomocou bloku **Outport**.

Pri analýze MATLAB funkcie bloku **Calibration** je dôležité vedieť, že sa funkcia spúšťa raz v každom výpočtovom kroku, čiže sa chová ako keby bola volaná v cykle s podmienkou trvania simulácie. Na začiatku funkcie sú deklarované premenné kľúčovým slovom **persistent**, ktoré sú lokálne pre danú funkciu, čiže zadržia svoju hodnotu pri viacnásobnom volaní funkcie. Použitie cyklov je nevhodné v tomto prípade a preto sú na cielené konanie príkazov používané iba jednoduché podmienky. V zásade, funkcia je rozdeľená na dve časti podmienkou, ktorá testuje hodnotu príznaku kalibrácie. V prípade že je zvolené vykonanie kalibrácie, algoritmus sa vetví na ďalšie podmienky, pričom kalibračné hodnoty sú zmerané a zapísané do premenných **Upper** a **Lower**. V opačnom prípade sa na preškálovanie použijú hodnoty zadané v maske. Zdrojový kód funkcie je uvedený v prílohe A.4.



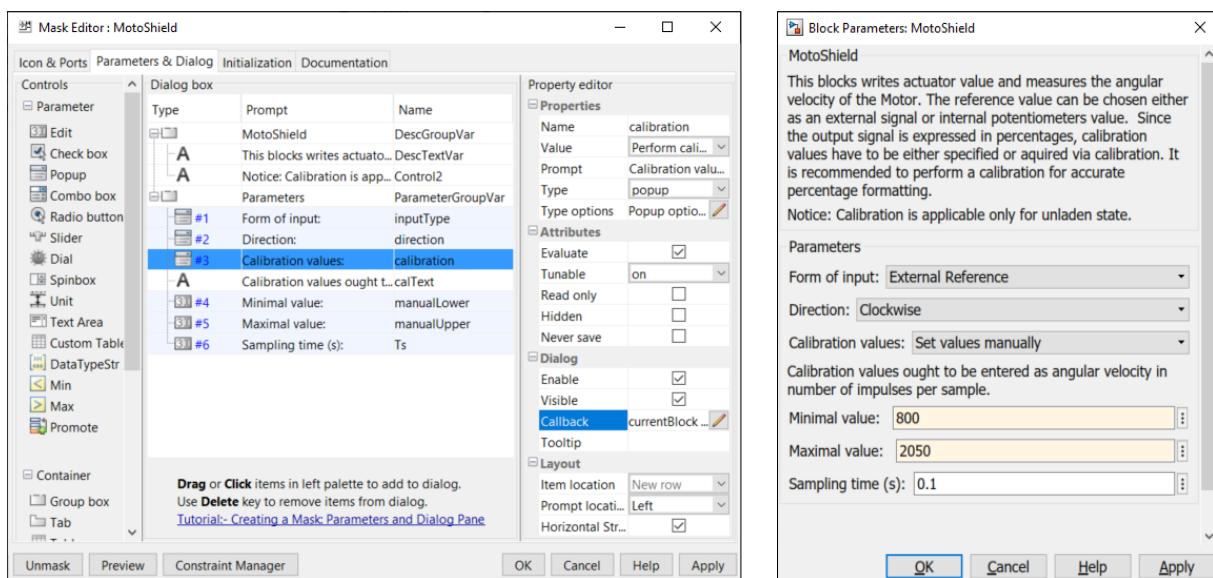
Obr. 2.6: Blok systému s externou referenčnou hodnotou – MotoShield

Maska

Maska je grafické okno podsystému, ktoré zobrazuje popis jeho funkčnosti a obsahuje interaktívne prvky, pomocou ktorých je možné meniť nastavenia, resp. jeho vnútornú štruktúru. Na tvorenie masky existuje grafické rozhranie tzv. „Mask Editor“. Toto rozhranie obsahuje štyri základné okná, z ktorých prvé je „Icon & Ports“. V tomto okne sa upravuje vzhľad ikonky bloku pomocou jednotlivých príkazov. Na zjednodušenie, pre všetky bloky bola nakreslená ikonka v softvéri vektorovej grafiky a nastavená k bloku nasledovným príkazom:

```
image('assets\MotoShield.svg');
```

Pomocou okna „Parameters & Dialog“ sa tvorí grafické rozhranie masky, čiže popis diaľkového okna a nastavenie interaktívnych prvkov. Interaktívny prvok, ktorý je použitý prvý v poradí je typu „Popup“ s označením **Form of input**. Je to v podstate padajúce menu s možnosťami **External Reference** a **Internal Reference**. Možnosti sú uvedené v okne **Properties** → **Type options**. Podľa toho, ktorá z možností padajúceho menu je zvolená, premenná **inputType** nadobudne hodnotu poradia zvolenej možnosti. Teda pri zvolení prvej možnosti v poradí **External Reference** premenná **inputType** nadobudne hodnotu 1. Názov premennej pre daný prvok je vlastne meno samotného prvku, ktoré je definované v stĺpci **Name**. Pomocou interaktívneho prvku **Form of input** je nastavovaný spôsob získania referenčnej hodnoty. V prípade zvolenia možnosti **External Reference** sa referenčná hodnota získa z bloku **Import**. V opačnom prípade sa referenčná hodnota získa z bloku **Reference Read**, pričom blok **Import** je odstránený z estetických dôvodov. Príkazy, pomocou ktorých sa takto zmení vnútorná štruktúra bloku sú napísané do okna „Initialization“ v rozhraní „Mask Editor“. Tieto príkazy sa vykonajú pri inicializácii bloku, čiže po kliknutí na tlačítko „Apply“ v maske daného bloku.



Obr. 2.7: Maska a Mask Editor bloku systému – MotoShield

```

set_param(gcb, 'MaskSelfModifiable', 'on');
if inputType == 1
    if getSimulinkBlockHandle([gcb '/Inport']) == -1
        delete_line(gcb,'Reference Read/1','Saturation/1');
        add_block('built-in/Inport',[gcb '/Inport']);
        add_line(gcb,'Inport/1','Saturation/1');
    end
elseif inputType == 2
    if getSimulinkBlockHandle([gcb '/Inport']) > 0
        delete_line(gcb,'Inport/1','Saturation/1');
        delete_block([gcb '/Inport']);
        add_line(gcb,'Reference Read/1','Saturation/1');
    end
end

```

Prvý príkaz povolí prístup na zmenu vlastného obsahu bloku. Príkaz `gcb` vráti cestu daného bloku v aktuálnom systéme. Potom nasleduje podmienka, ktorá testuje hodnotu premennej `inputType`, respektíve zvolenú možnosť padajúceho menu. Za tým je v rámci podmienky volaný príkaz `getSimulinkBlockHandle`, ktorého účelom je zistiť, či blok uvedený vo vstupnom argumente existuje. V prípade, že existuje, návratová hodnota funkcie bude nezáporná a v prípade že blok neexistuje, funkcia vráti hodnotu `-1`. Ak je zvolená možnosť `External Reference` a zároveň neexistuje blok `Inport`, príkazom `delete_line` sa vymaže spoj medzi blokmi `Reference Read` a `Saturation`. Za tým sa príkazom `add_block` pridá blok `Inport` s rovnakým názvom. Na koniec sa pomocou funkcie `add_line` pridá spoj medzi vytvoreným blokom a blokom `Saturation`. V prípade, že je zvolená možnosť `Internal Reference` a blok s názvom `Inport` existuje, blok sa spolu so spojom vymaže a vytvorí sa spoj s blokom `Reference Read`.

Druhý interaktívny prvok masky bloku `MotoShield` je rovnakým spôsobom použitý a zostavený aj pri maske bloku `Actuator Write`. Jedná sa o ďalší prvok typu „`Popup`“, čiže padajúce menu, ktorého účelom je umožniť nastavenie smeru otáčania mo-

tora prostredníctvom masky. Prvá možnosť prvku je pomenovaná **Clockwise** a druhá **Counterclockwise**. Meno prvku, čiže premennej, je **direction**. Premenné masky v rámci vnútornej schémy je možné volať pomocou bloku **Constant** s uvedením názvu premennej ako hodnoty. Schéma na nastavenie smeru otáčania je zobrazená na Obr. 2.3.

Tretí interaktívny prvok masky bloku **MotoShield** je pomenovaný **calibration** a používa okno **Dialog** → **Callback**. **Callback** je v podstate súbor príkazov, ktoré sa vykonajú pri každej zmene nastaviteľných parametrov daného prvku. Cieľom týchto príkazov je ihneď po voľbe jednej z možností upraviť vzhľad masky.

```

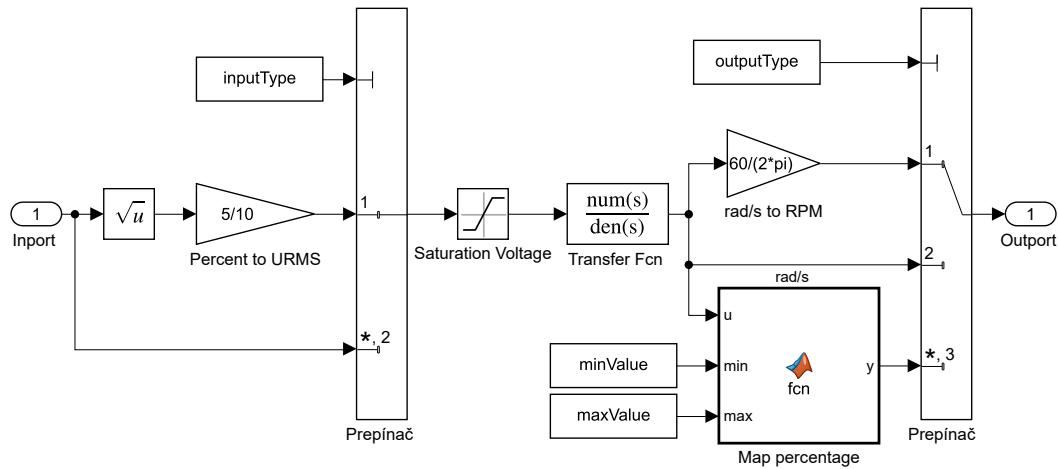
in = get_param(gcb,'calibration');
menu = Simulink.Mask.get(gcb);
txt = menu.getDialogControl('calText');
lower = menu.getParameter('manualLower');
upper = menu.getParameter('manualUpper');
if strcmp(in,'Perform calibration')
    lower.Enabled='off';
    lower.Visible='off';
    upper.Enabled='off';
    upper.Visible='off';
    txt.Visible='off';
elseif strcmp(in,'Use default values')
    lower.Enabled='off';
    lower.Visible='on';
    upper.Enabled='off';
    upper.Visible='on';
    set_param(gcb,'manualLower',num2str(800));
    set_param(gcb,'manualUpper',num2str(2050));
elseif strcmp(in,'Set values manually')
    lower.Enabled='on';
    lower.Visible='on';
    upper.Enabled='on';
    upper.Visible='on';
    txt.Visible='on';
end

```

V prvom riadku sa lokálnej premennej **in** priradí hodnota prvku **calibration**. Za tým sa vytvorí objekt masky daného bloku s názvom **menu**. Ďalej sa pomocou metód triedy **Simulink.Mask** získajú parametre ovládaných prvkov, ktoré sú menené v rámci podmienok. Podmienky testujú zhodnosť uvedeného reťazca s reťazcom premennej **in** pomocou funkcie **strcmp**. V prípade zhody funkcia vráti hodnotu 1 a v opačnom prípade hodnotu 0. Ovládané prvky **manualLower** a **manualUpper** sú typu „Edit“ a predstavujú textové polia do ktorých sa zapisujú kalibračné hodnoty. V prípade ľubo-volného definovania kalibračných hodnôt sú vlastnosti **Enabled** a **Visible** zapnuté. Pre možnosť voľby prednastavených hodnôt sa vlastnosť **Enabled** vypne, čo znemožní manipuláciu s poľom a súčasne sa hodnoty polí zmenia pomocou funkcie **set_param**. Samozrejme, v prípade vykonania kalibrácie, prvky na uvedenie kalibračných hodnôt budú vypnuté a neviditeľne. Posledný interaktívny prvok je taktiež typu „Edit“, pomenovaný **Ts**. Sem je potrebné uviesť hodnotu vzorkovacej periódy, ktorá sa potom prenesie dovnútra bloku **MotoShield**, do blokov **Digital Clock** a **Reference Read**.

2.3.4 Matematický model systému

V Kap. 4.2 je popísaná identifikácia systému tzv. „Greybox“ metódou, pričom medzi iným bola získaná aj prenosová funkcia. Systém bol identifikovaný z dát, pri ktorých vstupná veličina je efektívna hodnota napäťia U_{RMS} a výstupná uhlová rýchlosť $\dot{\theta}$ v jednotkách SI sústavy. Na poskytnutie možnosti iných jednotiek pre vstupné a výstupné veličiny bolo potrebné vytvoriť schému na preškálovanie.



Obr. 2.8: Blok prenosovej funkcie – MotoShield TF

V maske bloku MotoShield TF je na voľbu jednotky *vstupnej* veličiny uvedený interaktívny prvok `inputType`. Prvá možnosť v poradí sú percentá a druhá efektívna hodnota napäťia. Keďže prenosová funkcia na vstupe čaká efektívnu hodnotu napäťia, prvá vetva musí byť preškálovaná podľa vzťahu v Rov. (2.1), kym druhú vetvu priveďieme bez úprav. Pred vstupom do bloku prenosovej funkcie `Transfer Fcn` je hodnota ohraničená medzi 0 a 5 blokom pomenovaným `Saturation Voltage`.

Výstupný signál je dostupný v troch formách, pričom pridané jednotky sú percentá a otáčky za minútu. Prvok `outputType` je zodpovedný za voľbu jednotky výstupného signálu. Prvá možnosť interaktívneho prvku aktivuje prvú vetvu prepínača, kde sa signál z jednotky radián za sekundu násobkom $\frac{60}{2\pi}$ prevedie na jednotku otáčky za minútu. Pri tretej vetve je signál preškálovaný na percentá, pričom hodnoty `minValue` a `maxValue` sú hodnoty interaktívnych prvkov typu „Edit“ z masky bloku MotoShield TF.

3 Inštruktážne príklady

V nasledujúcej kapitole budú popísané inštruktážne príklady použitia vyhotovených programátorských rozhraní. Najprv budú prezentované príklady bez spätej väzby a v podkapitole PID na str. 30 budú prezentované príklady so spätnoväzbovým riadením.

3.1 Odozva v otvorenej slučke

Pre programátorské rozhranie Arduino IDE boli napísané dva inštruktážne príklady s otvorenou slučkou. Prvý je umiestnený v priečinku „examples\ MotoShield _StepResponse“ v repozitári AutomationShield na platforme GitHub [9]. Ako aj sám názov hovorí, ide o skokovú zmenu akčného zásahu pre ktorú platí podmienka v Rov. (3.1):

$$u(t) = \begin{cases} 100 & \text{pre } t \geq 0, \\ 0 & \text{pre } t < 0. \end{cases} \quad (3.1)$$

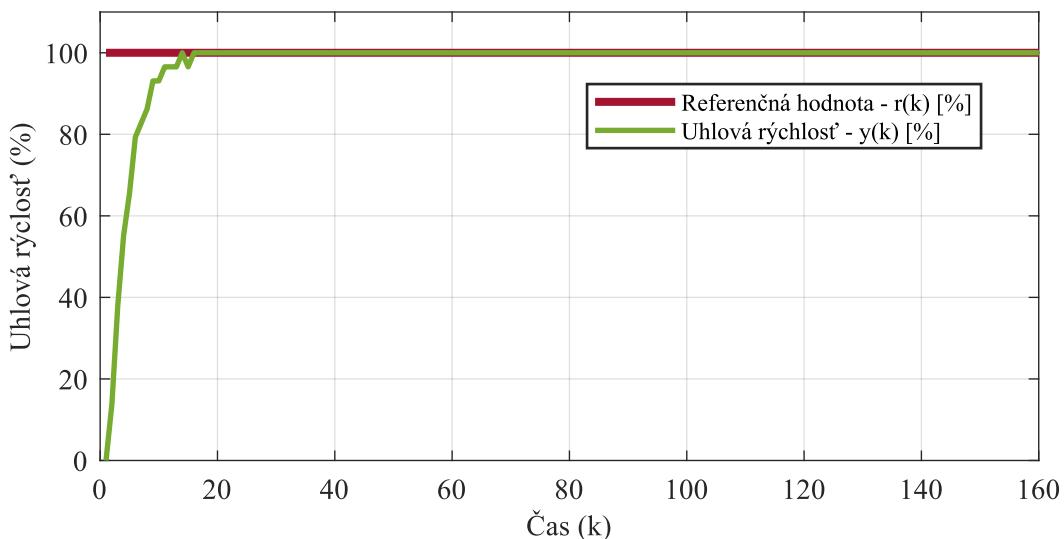
Pri použití programátorského rozhrania MotoShield je prvým krokom zahrnutie hlavičkového súboru preprocesorovou direktívou `#include < MotoShield.h >`. Potom sú v príklade definované symbolické konštanty vzorkovacej periódy a amplitúda skokovej zmeny pomocou preprocesorovej direktívy `#define`. Vývojové prostredie Arduino IDE má dve vstavané funkcie zamerané na organizáciu programu. Funkcia `void setup` sa vykoná iba raz po nahratí alebo reštartovaní programu a následne sa ustavične bude vykonávať funkcia `void loop`.

```
#include < MotoShield.h >
#define TS 20.0 //Vzorkovacia perióda
#define u 100.0 //Hodnota skokovej funkcie v percentách
void setup() {
    Serial.begin(250000);           //Inicializácia UART # 250000 bit/s
    MotoShield.begin(TS);          //Inicializácia MotoShieldu
    MotoShield.calibration();     //Kalibrácia
    MotoShield.actuatorWrite(u); //Akčný zásah
}
void loop() {
    Serial.print(MotoShield.sensorReadRPMPerc());
    Serial.print(" "); //Vysielanie informácií cez UART
    Serial.println(u);
}
```

Vo funkcií `void setup` je zavolaná inicializačná metóda `Serial.begin` pre perifériu UART¹ sériovej komunikácie mikroradiča, pomocou ktorej je realizovaná komunikácia

¹angl. Universal Asynchronous Receiver-Transmitter

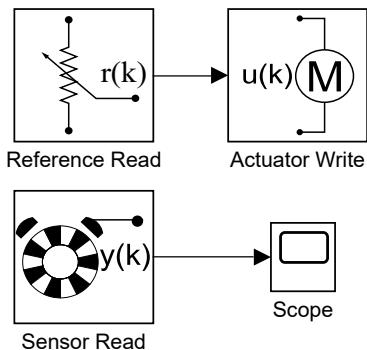
s počítačom. Taktiež je zavolaná aj metóda `MotoShield.begin(TS)` na inicializáciu modulu cez vopred vytvorený objekt `MotoShield` triedy `MotoShieldClass`. Pri inicializácii je špecifikovaná vzorkovacia períoda ako vstupný argument metódy. Potom je vykonaná kalibrácia, pomocou ktorej sa získajú kalibračné hodnoty a tak sa umožní meranie uhlovej rýchlosťi v percentách. Po vykonaní kalibrácie sa ihneď uskutoční zmena akčného zásahu metódou `MotoShield.actuatorWrite(u)` z 0% na 100%, ktorá vytrvá až do ukončenia behu programu. V neustále opakovanej časti programu sa bude cez sériový port posielat percentuálna hodnota uhlovej rýchlosťi a akčného zásahu. Toto je zabezpečené metódami `Serial.print()`, pričom sa najprv pošle uhlová rýchlosť získaná metódou `MotoShield.sensorReadRPMperc()`, potom medzera a na koniec hodnota akčného zásahu s ukončením riadku. Grafický výsledok príkladu „StepResponse“ je zobrazený na Obr. 3.1.



Obr. 3.1: Odozva systému MotoShield na skokovú zmenu

Druhý príklad, ktorý používa programátorské rozhranie napísané pre prostredie Arduino IDE je uložený v priečinku „examples\ MotoShield_OpenLoop“ v repozitári AutomationShield [9]. V porovnaní s predchádzajúcim príkladom, tento používa ešte aj metódu `referenceRead`. Základnou ideou príkladu je ovládať akčný zásah ručne, nastavením polohy bežca potenciometra.

Jednoduchý inštruktážny príklad bol vyhotovený aj pre prostredie Simulink, pomenovaný „InputsOutputs“. Idea je rovnaká ako aj pri príklade „OpenLoop“, vyčíta sa poloha bežca potenciometra, ktorá predstavuje akčný zásah. Súčasne sa meria uhlová rýchlosť motora a vypisuje cez tzv. `Scope`. Samozrejme, nastavenia jednotlivých blokov musia byť koherentné, totiž výstupný port z bloku `Reference Read` a vstupný port bloku `Actuator Write` musia byť v rovnakej stupnici. Blok sa nastavuje pomocou grafického okna, tzv. masky. V tomto príklade sú všetky bloky nastavené na percentuálnu stupnicu. Vzorkovaciu períodu majú bloky `Reference Read` a `Sensor Read` rovnakú, s hodnotou 0.02 sekúnd. Schéma príkladu je znázornená na Obr. 3.2



Obr. 3.2: Schéma inštruktážného príkladu InputsOutputs v prostredí Simulink

3.2 Spätnoväzbové riadenie a PID regulátor

Cieľom spätnoväzbového riadenia je odhadnúť akčný zásah, čiže hodnotu energie odozdanú akčnému členu tak, aby výstupná veličina systému nadobudla žiadanú hodnotu $r(t)$. Je to umožnené porovnávaním výstupnej $y(t)$ a vstupnej $u(t)$ hodnoty systému. Rozdiel týchto hodnôt v regulačnom obvode je tzv. regulačná odchýlka $e(t)$. Empirický bolo zistené, že ak motoru pridáme 55% vstupnej hodnoty, na výstupe dostaneme približne hodnotu 65% uhlovej rýchlosťi. Zmenou príkonu zistíme, že výstupná veličina je nepredvídateľne závislá od vstupnej. Preto je na roztočenie motora presnou uhlovou rýchlosťou potrebné navrhnuť regulačný obvod, ktorý sa bude snažiť vynulovať regulačnú odchýlku v reálnom čase. Spôsob ktorým je akčný zásah ovplyvnený regulačnou odchýlkou diktuje riadiaca jednotka.

PID alebo proporcionálne integračne derivačný regulátor je typ regulátora najpoužívanejší v technickej praxi [6]. Jeho názov predstavuje tri zložky z ktorých sa skladá:

- P zložka je proporcionálna časť regulátora. Výsledná hodnota P zložky je lineárne závislá od regulačnej odchýlky $e(t)$, pričom je smernica priamky definovaná proporcionálnou konštantou K_p . Proporcionálna konštanta je nastaviteľným parametrom pri návrhu regulátora. Pri zvolení príliš nízkej proporcionálnej konštanty sa regulovaná veličina $y(t)$ veľmi pomaly blíži k požadovanej hodnote $r(t)$. V opačnom prípade, ak je konštanta veľká, dochádza k rozkmitávaniu regulačného obvodu, čiže k nestabilite systému. Dôležité je si uvedomiť, že regulátor ktorý sa skladá iba z P zložky nedokáže vynulovať regulačnú odchýlku. Závislosť hodnoty akčného zásahu $u(t)$ od regulačnej odchýlky $e(t)$ v časovej oblasti je daná vzťahom v Rov. (3.2):

$$u(t) = K_p \cdot e(t). \quad (3.2)$$

- I zložka predstavuje integračnú časť regulátora. Výsledná hodnota I zložky je integrál regulačnej odchýlky počas doby regulácie, Rov. (3.3). V prípade, že je regulačná odchýlka kladná hodnota, integračná zložka navyšeje hodnotu akčného zásahu. V opačnom prípade I zložka bude hodnotu znižovať. Na rozdiel od P, integračná zložka je schopná vynulovať regulačnú odchýlku $e(t)$.

Integrovaná hodnota je násobená integračnou konštantou K_i , ktorá je nastaviteľným parametrom pri návrhu regulátora. Pri zvolení príliš vysokej hodnoty K_i regulovaná

veličina osciluje okolo požadovanej hodnoty:

$$u(t) = K_i \int_0^t e(\tau) d\tau . \quad (3.3)$$

- D zložka určuje rýchlosť zmeny regulačnej odchýlky. Pri nízkej rýchlosti zmeny regulačnej odchýlky derivačná zložka nemá veľký vplyv na hodnotu akčného zásahu. V prípade, že sa regulačná odchýlka začne rýchlo zmenšovať, derivačná zložka bude zmenšovať akčný zásah. Kvôli týmto vlastnostiam zmenšuje prekmitnutie pri regulačnom procese. Taktiež je násobená nastaviteľným parametrom, derivačnou konštantou K_d . Zvýšením hodnoty K_d je regulačný proces pomalší, ale systém je stabilnejší:

$$u(t) = K_d \frac{de(t)}{dt} . \quad (3.4)$$

Súčtom všetkých troch zložiek dostaneme závislosť akčného zásahu od regulačnej odchýlky podľa PID regulátora. Rovnica (3.5) predstavuje paralelný zápis PID v časovej oblasti:

$$u(t) = K_p \cdot e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt} . \quad (3.5)$$

V praxi sa často stretнем aj s tzv. ideálnou formou PID regulátora Rov. (3.6). Na rozdiel od paralelného, ideálny má iba jednu konštantu zosilnenia K_p , ktorá ovplyvňuje všetky tri zložky regulátora. Integračná a derivačná zložka sú individuálne nastaviteľné pomocou integračnej T_i a derivačnej T_d časovej konštanty:

$$u(t) = K_p \left(e(t) + \frac{1}{T_i} \int_0^t e(\tau) d\tau + T_d \frac{de(t)}{dt} \right) . \quad (3.6)$$

Regulačné obvody, pri ktorých aspoň jeden člen pracuje diskrétnie v čase, sa nazývajú diskrétné regulačné obvody. Výskyt diskrétnych regulačných obvodov je najčastejšie dôsledkom použitia diskrétnych regulátorov - počítačov. Taktiež sa vyskytujú aj v prípadoch, keď regulovaná veličina nemôže byť meraná spojito [6]. Pri regulačnom systéme pre MotoShield obidva dôvody platia, a preto je v príkladoch riadenia použitý diskrétny PID regulátor (3.7).

Transformácia zo spojitej časovej oblasti do diskrétnej spočíva v rovnomenom podelení času t na indexované celky kT , tzv. vzorky. Vzorkovacia períoda T je časový interval medzi dvomi po sebe idúcimi vzorkami. Diskrétny PID regulátor v tzv. ideálnej podobe je vyjadrený nasledovne:

$$u(kT) = K_p \left(e(kT) + \frac{T}{T_i} \sum_{i=0}^k e(iT) + \frac{Td}{T} (e(kT) - e((k-1)T)) \right) \mid k \in N \cup \{0\} . \quad (3.7)$$

3.3 Príklady riadenia pomocou PID regulátora

V nasledujúcej podkapitole sú popísané príklady na spätnoväzbové riadenie uhlovej rýchlosťi motora pomocou PID regulátora.

3.3.1 Arduino IDE

V príklade riadenia v prostredí Arduino IDE je zahrnutá knižnica² príkazom `#include <PIDAbs.h>`, ktorá zahŕňa algoritmus ideálneho PID regulátora s obmedzením integračného nasýtenia³ (angl. anti-windup). Nastaviteľné parametre regulátora, K_p , T_i a T_d , sú definované ako symbolické konštanty preprocesorom a k samotnému PID algoritmu sú priradené pomocou metód `PIDAbs::set*`. Vzorkovacia períoda T_s je priradená aj riadiacemu algoritmu, aj inicializačnej metóde hardvéru. Inicializácia sériovej komunikácie a MotoShieldu, spolu s kalibráciou a priradením nastaviteľných hodnôt regulátora, sa vykoná v rámci organizačnej funkcie `setup`.

```
#define TS 40.0          // Vzorkovacia períoda
#define AUTO 1            // Režim nastavenia referencie
#define KP 0.000001        // Parametre regulátora
#define TI 0.0003
#define TD 0.001
float R[]={40.0,70.0,50.0,85.0,35.0,90.0,50.0,70.0}; //Trajektória referenčnej
→ hodnoty
unsigned int k = 0; //Index vzorky
int T = 80;          //Dĺžka sekcie
int i = 0;           //Index referenčnej hodnoty
void setup() {
    Serial.begin(2000000); //Inicializácia UART
    MotoShield.begin(TS); //Inicializácia modulu MotoShield
    MotoShield.calibration(); //Kalibrácia
    PIDAbs.setKp(KP); // Priradenie parametrov
    PIDAbs.setTi(TI);
    PIDAbs.setTd(TD);
    PIDAbs.setTs(TS);
}
```

V organizačnej funkcií `loop` je v rámci vzorkovacieho algoritmu volaná funkcia `step`. Testovaná premenná `MotoShieldClass::stepEnable` nadobudne kladnú hodnotu na konci každej vzorky⁴, podmienka sa splní, vykoná sa funkcia `step` a na konci sa premennej priradí nulová hodnota.

```
void loop() {
    if (MotoShieldClass::stepEnable) {
        step();
        MotoShieldClass::stepEnable=false;
    }
}
```

Funkcia `step` môže nadobúdať dva rôzne tvary v závislosti od zvoleného režimu nastavenia požadovanej hodnoty. Režim sa volí definíciou symbolickej konštanty `AUTO`. V prípade, že konštantu definujeme ako nulovú, požadovaná hodnota bude nastavovaná bežcom potenciometra. V opačnom prípade požadovaná hodnota bude definovaná množinou `R`, pričom sa prvok množiny volí pomocou indexu `i`. Podmienka na navýšenie indexu `i` je nulový zvyšok pri delení indexu vzorky `k` hodnotou `T*i`. Premenná `T` symbolizuje počet vzoriek pre zmenu referenčnej hodnoty.

²Vytvorená v rámci projektu AutomationShield [9].

³K nasýteniu integračnej zložky dochádza v prípade že akčný člen je neschopný dosiahnuť požadovanú hodnotu.

⁴Toto je zabezpečené v rámci obsluhy cyklického prerušenia, ktorú nájdete na str. 13

Príkazy na meranie regulovanej veličiny, výpočet akčného zásahu a jeho vyslanie sú rovnaké bez ohľadu na režim nastavenia požadovanej hodnoty. Hodnota akčného zásahu je vypočítaná príkazom `PIDAbs.compute(r - y, 0, 100, 0, 100)`, pričom vstupný argument `r - y` predstavuje regulačnú odchýlku. Druhý a tretí argument reprezentujú saturáciu akčného člena a štvrtý s piatym sú hranice integračnej zložky, tzv. anti-windup. Na konci funkcie `step` sú volané metódy sériovej komunikácie na vyslanie referenčnej hodnoty a nameraného výstupu.

```
void step(){
#if !AUTO
    r = MotoShield.referenceRead();
#else AUTO
    if(i >= sizeof(R)/sizeof(float)){//Ak je index trajektórie väčší ako veľkosť
        → množiny R
        MotoShield.actuatorWrite(0.0); // Zastav motor
        while(true); // Ukončenie programu
    }
    if (k % (T*i) == 0){ // Pokračovanie v trajektórii
        r = R[i];
        i++;
    }
    k++;
#endif
    y = MotoShield.sensorReadRPMperc();
    u = PIDAbs.compute(r-y,0,100,0,100);
    MotoShield.actuatorWrite(u);
    Serial.print(r);
    Serial.print(" ");
    Serial.println(y);
}
```

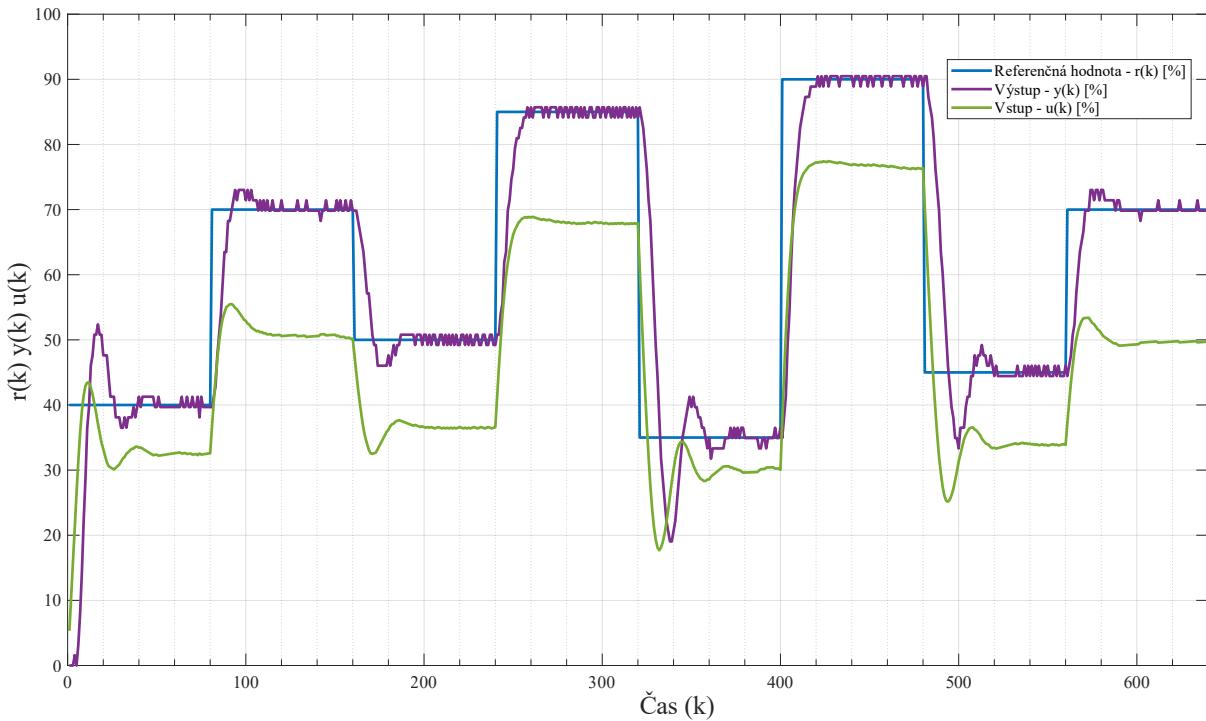
Grafický výstup príkladu pri uvedených parametroch je znázornený na Obr. (3.3). Príklad v celosti nájdete v prílohe B.

3.3.2 MATLAB

Ak porovnáme príklady PID riadenia v prostredí Arduino IDE a MATLAB zistíme, že existuje množstvo rozdielov. Rozdiely spočívajú hlavne v spôsobe použitia programátor-ských rozhraní. Na začiatku je potrebné definovať režim nastavenia referenčnej hodnoty. Potom sa vytvorí objekt (inštancia) `MotoShield` triedy `MotoShield`. Za tým nasleduje inicializačná metóda hardvéru `begin`, ktorá si vyžaduje uvedenie modelu použitej prototypizačnej dosky a číslo sériového (COM) portu, cez ktorý je k počítaču pripojená. Ďalej sa prostredníctvom vytvoreného objektu volá kalibračná funkcia, ktorá je povinná v prípade vyjadrenia meranej veličiny v percentách.

```
Manual = 0; %Definovanie režimu nastavenia ref. hodnoty
MotoShield = MotoShield;%Tvorenie objektu
MotoShield.begin('COM4','UNO');
MotoShield.calibration()
```

V ďalšej časti kódu je vytvorený objekt triedy `PID` s rovnakým názvom, ktorý disponuje metódou PID výpočtu `compute`. Nastaviteľné parametre sú regulátoru priradené príkazom `PID.setParameters(Kp, Ti, Td, Ts)`.



Obr. 3.3: Výsledok PID riadenia v prostredí Arduino IDE

```
PID = PID; %Vytvorenie objektu
Ts = 0.04; %Definovanie parametrov regulátora
Kp = 0.05;
Ti = 0.003;
Td = 1;
PID.setParameters(Kp, Ti, Td, Ts); %Priradenie parametrov regulátoru
```

Za tým nasleduje definovanie pomocných premenných, ako sú trajektória referenčných hodnôt, index vzoriek, index trajektórie atď.

```
k = 1; %Index vzorky
stepEnable = 0; %Príznak vzorkovacieho algoritmu
R = [80, 70, 50, 85, 35, 60]; %Trajektória referenčnej hodnoty
T = 100; %Dĺžka sekcie
i = 0; %Index trajektórie
```

Keďže v rámci programátorského prostredia MATLAB nie je použité cyklické prerušenie, na vytvorenie vzorkovacieho algoritmu potrebujeme merať čas. V prostredí MATLAB, existujú dva príkazy pomocou ktorých sa čas merá. Na začiatie merania času sa zavolá príkaz `tic` a na vyčítanie aktuálneho času príkaz `toc`. Vzorkovací algoritmus, meranie regulovanej veličiny, zmenu akčného zásahu a pod. je potrebné vykonávať ustavične počas behu programu. V prostredí Arduino IDE je nekonečný cyklus zabezpečený organizačnou funkciou `loop`, ktorú v MATLAB-e nahradíme príkazom `while (1)`.

Vzorkovací algoritmus spočíva v podmienke, ktorá je splnená v prípade, že aktuálny čas nadobudne hodnotu väčšiu ako je kumulatívna hodnota nasledovnej vzorky. Kumulatívna hodnota nasledovnej vzorky sa rovná násobku hodnoty vzorkovacej períody s hodnotou indexu nasledovnej vzorky $Ts * k$. V prípade splnenia podmienky príznak povolenia výpočtového algoritmu `stepEnable` nadobudne kladnú hodnotu. Pripomeňme, že takéto

vzorkovanie v skriptovacom jazyku MATLAB nezabezpečuje riadenie striktne v reálnom čase. Avšak pre didaktické účely je postačujúce.

```

tic                                %Začiatok merania času
while (1)                          %Nekonečný cyklus
    if (toc >= Ts * k)            %Vzorkovacia podmienka
        stepEnable = 1;           %Povolenie algoritmu
    end

```

Ďalej nasleduje niekoľko podmienok, kde prvá testuje hodnotu príznaku `stepEnable`. V prípade že je podmienka splnená, nasledujú podmienky na zistenie režimu nastavenia referenčnej hodnoty. Ak je premenná `Manual` definovaná ako nulová, referenčné hodnoty budú volené z množiny `R`. Podmienka `if (mod(k, T*i) == 1)` zistuje, či aktuálna vzorka patrí do ďalšej sekcie žiadanej hodnoty. V prípade, že patrí, index trajektórie `i` bude zvýšený o hodnotu 1 a zvolí sa ďalší prvok množiny `R` za referenčnú hodnotu, príkazom `r=R(i)`. V prípade, že index trajektórie `i` je väčší ako počet prvkov množiny `R`, motor sa vypne, nekonečný cyklus sa ukončí príkazom `break` a zároveň sa ukončí aj spúštanie programu.

```

if (stepEnable)                      %Povolenie spustenia algoritmu
    if(Manual == 0)                  %Automatický režim referencie
        if (mod(k, T*i) == 1)        %Zmena ref hodnoty
            i = i + 1;
            r = R(i);                %Progresia v trajektórii
            if (i > length(R))
                MotoShield.actuatorWrite(0.0); %Vypnutie motora
                break                   %Ukončenie nekonečného cyklu
            end
        end
    end
    if(Manual == 1)
        r = MotoShield.referenceRead(); %Čítanie polohy bežca
    end

```

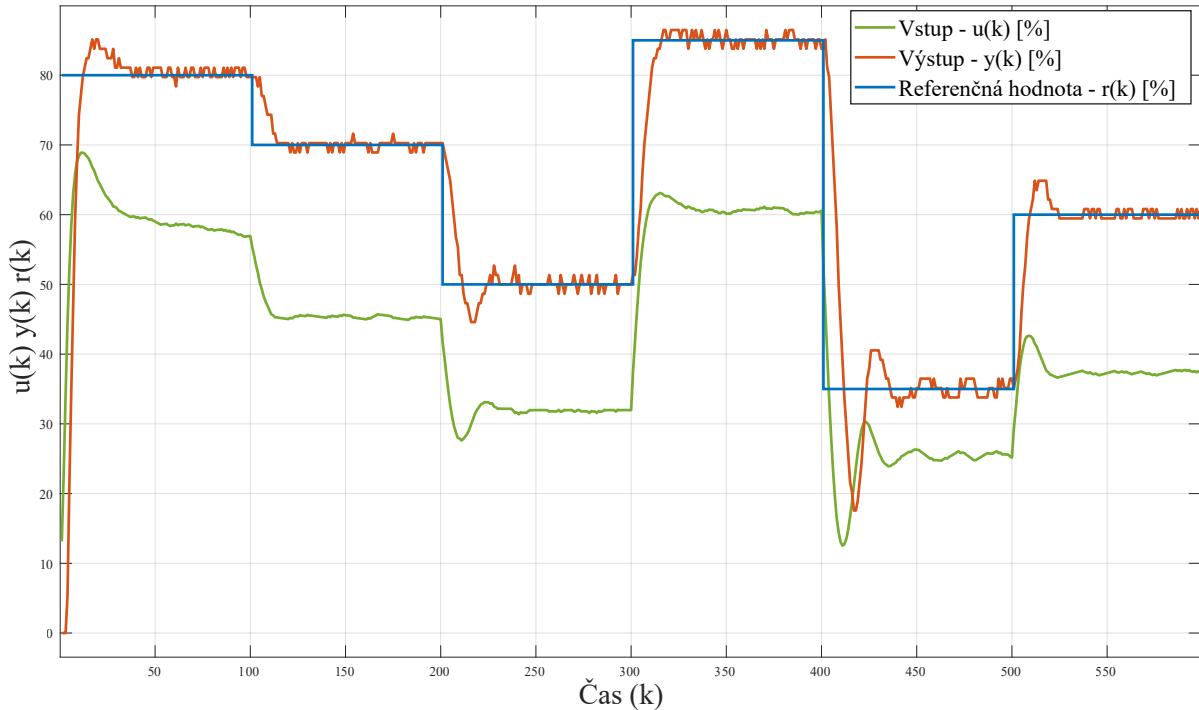
Po nastavení referenčnej hodnoty nasleduje príkaz na meranie regulovanej veličiny, výpočet akčného zásahu PID algoritmom a zápis akčného zásahu. Výpočet akčného zásahu je umožnení metódou `compute(r-y, 0, 100, 0, 100)` triedy `PID`, kde `r-y` predstavujú regulačnú odchýlku. Záznam hodnôt regulačného obvodu je zapisovaný do matice `response` rozmerov $k \times 3$. Na konci algoritmu je index vzorky `k` inrementovaný a spustenie algoritmu je zamietnuté vynulovaním premennej `stepEnable`.

```

y = MotoShield.sensorReadRPMperc();   %Meranie uhlovej rýchlosťi
u = PID.compute(r-y, 0, 100, 0, 100); %PID výpočet akč. zásahu
MotoShield.actuatorWrite(u);          %Zápis akčného zásahu
response(k, :) = [r, y, u];          %Záznam
k = k + 1;                           %Inkrement indexu vzorky
stepEnable = 0;                      %Znemožnenie spustenia algoritmu
end                                   %Koniec if(stepEnable)
end                                   %Koniec while 1

```

Na konci príkladu je záznam hodnôt exportovaný do súboru `response.mat`, ktorý je potom príkazom `plotPIDResponse` grafický zobrazený. Výsledok PID riadenia s uvedenými parametrami je znázornený na Obr. 3.4.



Obr. 3.4: Výsledok PID riadenia v prostredí MATLAB

3.3.3 Simulink

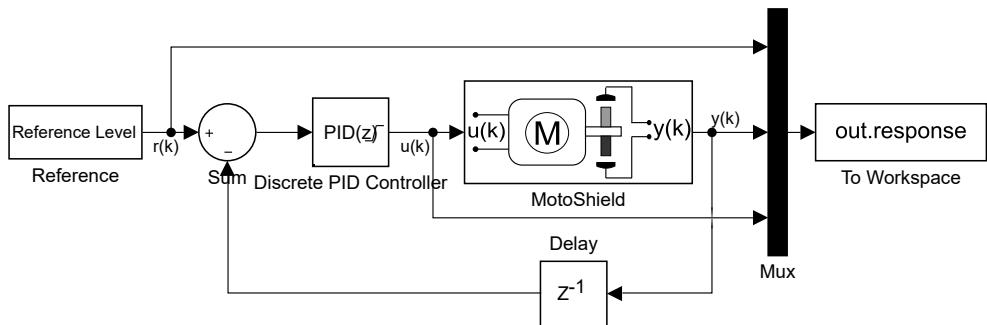
Príklad riadenia pomocou PID regulátora v prostredí Simulink je omnoho jednoduchší v porovnaní s príkladmi pre MATLAB a Arduino IDE. Blok **MotoShield** predstavuje regulovanú sústavu, pre ktorú je aj vstupná, aj výstupná veličina vyjadrená v percentách. Kalibračné hodnoty sú získané vykonaním kalibrácie a vzorkovacia periódna má hodnotu rovnakú ako aj v minulých príkladoch $T_s = 0.04$ s. Na nastavenie referenčnej hodnoty je použitý blok **Reference**, vytvorený v rámci projektu AutomationShield [9]. Referenčné úrovne sú nastavené v maske, uvedené v textovom poli s názvom **Reference vector**. Hodnoty pre tento konkrétny príklad sú $[80, 40, 90, 50, 70, 55, 65]$ (%). V poli **Section length** je definovaný počet vzoriek (70) pre jednu referenčnú úroveň. Vzorkovacia periódna je rovnaká, ako aj pri bloku **MotoShield**. Keďže sa pri spustení programu najprv vykoná kalibrácia ktorá trvá určitú dobu, v textovom poli **Offset time at start** je definované oneskorenie 8 sekúnd.

Blok **Discrete PID Controller** predstavuje PID algoritmus. Pomocou masky je nastavený ako diskrétny vo forme **Ideal**. Prenosová funkcia regulátora je znázornená v Rov. (3.8), kde konštantá P má hodnotu 1, I 4.5 a D 0.01.

$$R(z) = P \left(1 + I \cdot T_s \frac{1}{z-1} + D \cdot \frac{1}{T_s} \frac{z-1}{z} \right) \quad (3.8)$$

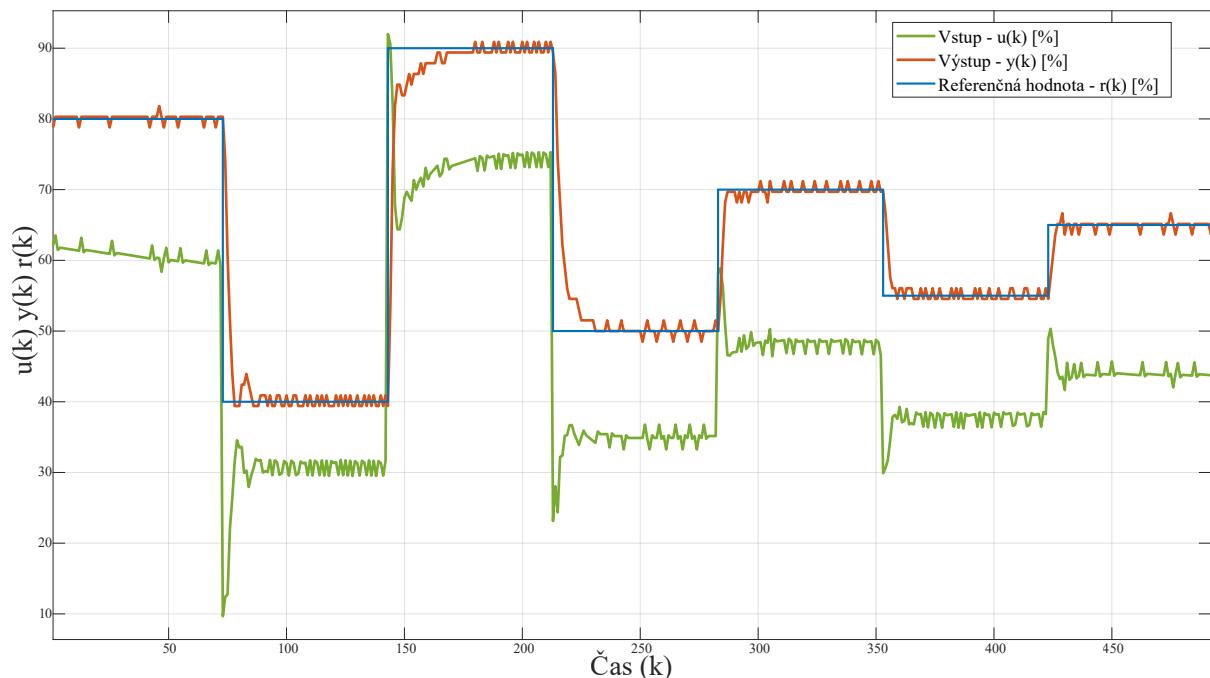
V okne **Output Saturation** sú nastavené hranice obmedzenia (100 a 0) výstupnej hodnoty. Taktiež je tu zvolené aj anti-windup zabezpečenie v režime **clamping**. Vzorkovacia periódna je nastavená na hodnotu -1 , čo znamená, že blok periódu zdedí zo vstupného signálu.

Blok **Delay** predstavuje oneskorenie o jednu hodnotu, čiže o jednu vzorku. Hodnoty



Obr. 3.5: Bloková schéma PID riadenia v prostredí Simulink

signálov $r(k)$, $u(k)$ a $y(k)$ sú zapisované do matice **response** v prostredí MATLAB. V MATLAB-e sú najprv z matice odstránené hodnoty zozbierané počas kalibrácie, a potom je matice exportovaná do súboru **response.mat**. Grafické zobrazenie dát je predvedené funkciou **plotPIDResponse**.



Obr. 3.6: Výsledok PID riadenia v prostredí Simulink

4 Modelovanie a identifikácia

Súvislosť medzi vstupnými a výstupnými veličinami a dynamické správanie sa systému je možné popísať matematicko-fyzikálnymi rovnicami. Proces odvodzovania týchto rovníc sa volá *matematické modelovanie*, ktorého výsledkom je *matematický model*. Pri odvodzovaní rovníc vychádzame zo základných znalostí a axiom fyziky pre konkrétny modelovaný systém. Odvodený model je všeobecný pre systém daného typu, čiže neobsahuje žiadne kvantitatívne informácie o reálnom systéme. V prípade že nie sú časovo premenné, kvantitatívne informácie (charakteristiky) systému sú definované jednotlivými konštantami. Tieto konštanty možno získať meraním alebo procesom *experimentálnej identifikácie*, ktorý vychádza zo vstupno-výstupných údajov systému.

4.1 Tvorba modelu

Jednosmerný elektrický motor je zariadenie schopné previesť elektrickú energiu na mechanickú a naopak. Preto matematický model jednosmerného motora vychádza z rovnice, ktorá popisuje dynamiku mechanických veličín a ďalšej rovnice, ktorá predstaví dynamiku elektromagnetických javov.

Pre popis mechanických vlastností budeme vychádzať z druhého Newtonovho zákona pre rotačný pohyb, Rov.(4.1), kde J je moment zotrvačnosti, $\ddot{\theta}$ je uhlové zrýchlenie a M je krútiaci moment. Tretí moment M_t pôsobí proti smeru otáčania a preto v je Rov. (4.2) záporný. Taktiež je lineárne závislý od uhlovej rýchlosťi, pričom koeficient viskózneho tlmenia b predstavuje smernicu. Hnací moment M_h spôsobuje rotačný pohyb hriadeľa a jeho hodnota vo všeobecnosti je proporcionalna prúdu a intenzite magnetického poľa. Za predpokladu, že magnetické pole je konštantné, hnací moment bude lineárne úmerný prúdu i podľa koeficientu momentu motora K_t .

$$J\ddot{\theta} = \sum_x \vec{M}_x = \vec{M}_h + \vec{M}_t \quad (4.1)$$

$$J\ddot{\theta} = M_h - M_t = K_t i - b\dot{\theta} \quad (4.2)$$

Pri rovnici dynamiky elektrického obvodu, Rov. (4.3), vychádzame z Kirchhoffovho druhého zákona. Zákon hovorí, že *súčet úbytkov napäťia na spotrebičoch je rovný súčtu napäti zdrojov v uzavretej časti obvodu*[11]. V prípade jednosmerného motora je úbytok napäťia spôsobený celkovým odporom obvodu R , indukčnosťou cievky L a elektromotorickou silou \mathcal{E} . Elektromotorická sila je v zásade generované napätie prácou neelektrických sôl. Pri jednosmernom motore elektromotorická sila priamo úmerne závisí od rýchlosťi otáčania hriadeľa $\dot{\theta}$. Pomer medzi rýchlosťou hriadeľa a indukovaným napäťím je daný

koeficientom elektromotorickej sily K_e :

$$U_z = Ri + L \frac{di}{dt} + \mathcal{E} = Ri + L \frac{di}{dt} + K_e \dot{\theta}. \quad (4.3)$$

Ak koeficienty K_t a K_e vyjadrieme v jednotkach SI sústavy, ich hodnota, ktorú označíme symbolom K , bude rovnaká:

$$K_t = K_e = K. \quad (4.4)$$

4.1.1 Prenosová funkcia

V rovniach (4.2) - (4.3) považujeme prúd i a uhlovú rýchlosť $\dot{\theta}$ za stavové veličiny systému. Ak použijeme tzv. SISO¹ prenosovú funkciu, schopný sme vyjadriť závislosť jednej výstupnej a jednej vstupnej veličiny, konkrétnie závislosť uhlovej rýchlosťi $\dot{\theta}$ a vstupného napäťa U_z . Po aplikovaní Rov. (4.4) a Laplaceovej transformácii na diferenciálne rovnice (4.2) - (4.3) dostaneme Rov. (4.5) - (4.6), ktoré sú vyjadrené v obrazovej oblasti:

$$Js^2\Theta(s) = KI(s) - bs\Theta(s) \implies I(s) = s\Theta(s) \frac{(Js + b)}{K}, \quad (4.5)$$

$$U_z(s) = RI(s) + LsI(s) + Ks\Theta(s) = I(s)(R + Ls) + Ks\Theta(s), \quad (4.6)$$

a po substitúcii dostaneme prenosovú funkciu, čiže prenos $G(s)$:

$$G(s) = \frac{s\Theta(s)}{U_z(s)} = \frac{K}{(K^2 + (Js + b)(Ls + R))}. \quad (4.7)$$

4.1.2 Stavový priestor

Ďalšia forma matematického modelu je stavová reprezentácia. Charakteristická je tým, že systém je predstavený skupinou diferenciálnych rovníc prvého rádu popisujúce dynamiku stavových veličín a pomocou skupiny algebraických rovníc, ktoré definujú výstup systému [18]. Používa sa hlavne pre MIMO² systémy. Všeobecný popis stavovej reprezentácie z Rov. (4.8) hovorí, že zmeny stavov \dot{x} a výstupy y sú funkcie stavov x a vstupov u do systému.

$$\begin{aligned} \dot{x}_1 &= f_1(x_1, \dots, x_i, u_1, \dots, u_j) & y_1 &= g_1(x_1, \dots, x_i, u_1, \dots, u_j) \\ &\vdots & &\vdots \\ \dot{x}_i &= \underbrace{f_i(x_1, \dots, x_i, u_1, \dots, u_j)}_{\text{Zmena stavov}} & y_n &= \underbrace{g_n(x_1, \dots, x_i, u_1, \dots, u_j)}_{\text{Výstupy}} \end{aligned} \quad (4.8)$$

V prípade, že všetky funkcie výstupov a zmeny stavov sú lineárne, model je lineárny a dá sa zapísť pomocou matíc a vektorov, Rov. (4.9). Matice systému **A** definuje zmenu stavových veličín od aktuálneho stavu systému, ktoré matice **B** definuje zmenu stavových veličín v závislosti od vstupných veličín. Vektor výstupov \vec{y} závisí od stavu systému podľa

¹abbrev. Single-Input Single-Output.

²abbrev. Multiple-Input Multiple-Output.

matice **C**. Matica **B** je nenulová iba v prípade, ak niektorá zo vstupných veličín priamo ovplyvňuje stav systému:

$$\begin{aligned}\dot{\vec{x}} &= \mathbf{A}\vec{x} + \mathbf{B}\vec{u}, \\ \vec{y} &= \mathbf{C}\vec{x} + \mathbf{D}\vec{u}.\end{aligned}\tag{4.9}$$

Keďže stavová rovnica vychádza z rovníc dynamiky systému (4.3) a (4.2), rovnice upravíme tak, aby vyjadrovali zmenu stavových veličín:

$$\begin{aligned}\frac{d\dot{\theta}}{dt} &= \frac{K}{J}i - \frac{b}{J}\dot{\theta}, \\ \frac{di}{dt} &= -\frac{R}{L}i - \frac{K}{L}\dot{\theta} + \frac{1}{L}U_z.\end{aligned}$$

Stavová rovnica modelu motoru bude mať nasledovný tvar:

$$\frac{d}{dt} \begin{bmatrix} \dot{\theta} \\ i \end{bmatrix} = \begin{bmatrix} -\frac{b}{J} & \frac{K}{J} \\ -\frac{K}{L} & -\frac{R}{L} \end{bmatrix} \begin{bmatrix} \dot{\theta} \\ i \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{L} \end{bmatrix} U_z.\tag{4.10}$$

Keďže chceme merať obe stavové veličiny, musia byť prehlásené za výstupné. Toto sa docieli zapísaním rovníc $y_1 = \dot{\theta}$ a $y_2 = i$ do maticovej podoby vo forme výstupovej rovnici stavového modelu. Samozrejme, matica **D** zostane nulová, lebo vstupná veličina priamo neovplyvňuje výstup:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{\theta} \\ i \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} U_z.\tag{4.11}$$

4.2 Identifikácia

Parametre analyticky odvodeného modelu prenosovej funkcie a stavového modelu získame pomocou experimentálnej identifikácie. Takýto typ identifikácie využije údaje výstupných veličín nameraných počas cieleného experimentu zberu dát.

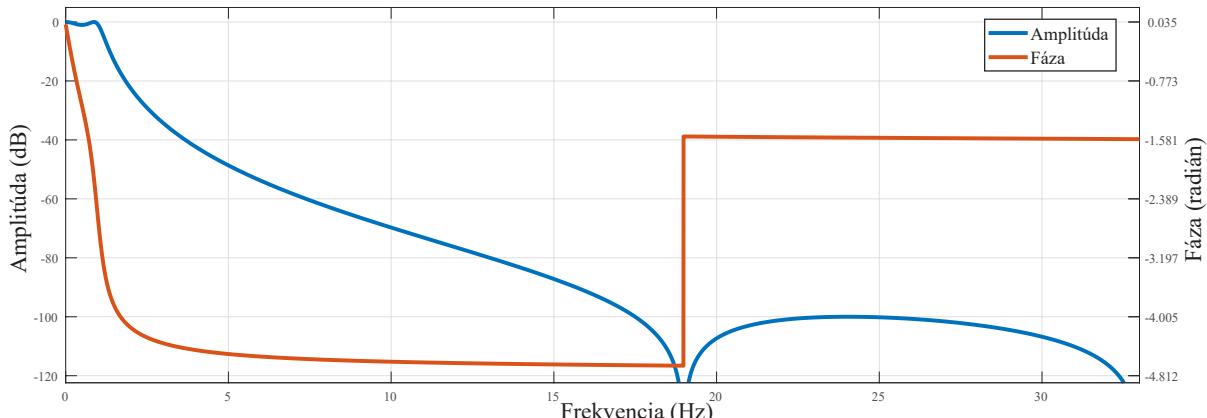
Úspešnosť experimentu veľmi závisí od typu vstupného testovacieho signálu. Pre tento experiment je vstupný signál pseudonáhodný, viachladinový, vygenerovaný prostredníctvom MATLAB funkcie `aprbsgenerate`³. Pri generovaní signálu boli definované hranice potenciálnych hodnôt a frekvenčná charakteristika signálu. Vygenerovaný signál je zachovaný vo forme hlavičkového súboru, pričom hodnoty signálu sú zapísané do množiny `aprbssU`.

Zber dát bol vykonaný pomerne jednoduchým zdrojovým kódom súboru „MotoShield_Identification“. V zdrojovom kóde sa najprv vykoná zahrnutie hlavičkového súboru vygenerovaného testovacieho signálu, pričom sa množina hodnôt signálu zapíše do programovej pamäte. Potom sa zahrnie programátorské rozhranie prístroja a zároveň sa

³Funkcia bola napísaná v rámci projektu AutomationShield [9].

vykoná inicializácia hardvéru a sériového rozhrania. V organizačnej funkcií `loop` je volaný algoritmus `step` v rámci vzorkovacej podmienky, rovnako ako v príklade PID riadenia. Algoritmus `step` pomocou príkazu `pgm_read_float_near(&aprbsU[i])` číta hodnotu signálu, pričom argument `&aprbsU[i]` symbolizuje miesto v pamäti pre danú hodnotu. Hodnota testovacieho signálu je vyslaná akčnému členu metódou `MotoShield.actuatorWriteVolt`. Záznam výstupných veličín je počítaču posielaný prostredníctvom sériového rozhrania. Uhlová rýchlosť je meraná v jednotke počet impulzov na vzorku, čo je vlastne hodnotou premennej `MotoShieldClass::counted`. Prúd je meraný priamo v jednotke SI sústavy metódou `MotoShield.sensorReadCurrent`.

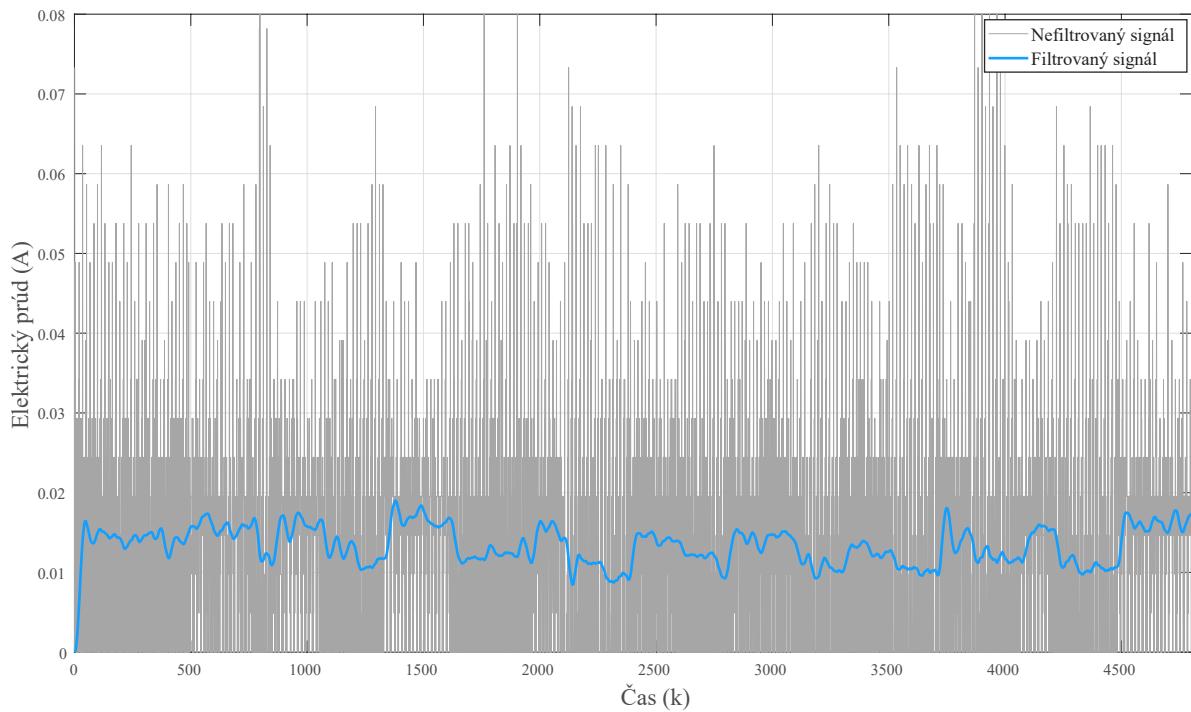
Namerané dátá boli upravené v prostredí MATLAB. Keďže všetky veličiny musia byť vyjadrené v jednotkách SI sústavy, nameraná uhlová rýchlosť bola prevedená na jednotky radián za sekundu násobkom $\frac{2\pi}{14T_s}$. Vzorkovacia periódka T_s v tomto experimente mala hodnotu 0.02 sekúnd. Po grafickom zobrazení nameraných dát odberu prúdu je zrejmé, že signál prúdu je veľmi zašumený, a preto nie je vhodný pre algoritmy identifikácie. Na útlm šumu bol použitý číslicový hornopriepustný filter s nekonečnou impulzovou odozvou (angl. infinite impulse response, IIR). Filter bol navrhnutý v grafickom rozhraní „Filter Designer“, ktoré je súčasťou balíku *Signal Processing Toolbox*. Navrhnutý filter je eliptický, tretieho rádu. Amplitúdová a fázová charakteristika filtra je graficky zobrazená na Obr. 4.1. Vytvorený filter z rozhrania „Filter Designer“ bol exportovaný do pracovného priestoru



Obr. 4.1: Amplitúdová a frekvenčná charakteristika hornopriepustného filtrovania

vo forme objektu pomenovaného `Hd`. Zašumený signál bol prepustený cez filter príkazom `filter(Hd, i)`. Výsledok filtrovania je vykreslený na Obr. 4.2.

Identifikácia bola uskutočnená v prostredí MATLAB pomocou balíka *System Identification Toolbox*. Program pre identifikáciu je pre prenosovú funkciu a stavový priestor písaný zvlášť. Logika je pri obidvoch rovnaká. Najprv sa vytvorí dátový objekt príkazom `iddata`. Tento objekt obsahuje vstupné, výstupné dátá a vzorkovaciu periódnu. Pri prenosovej funkcií je výstupná veličina iba uhlová rýchlosť, kým pri stavovom priestore je výstupom aj elektrický prúd. Potom sa vynuluje trend dát príkazom `detrend`. Ďalším krokom je vytvoriť začiatočný odhad identifikovaného modelu. Pri definovaní hodnôt konštánt začiatočného modelu bol zmeraný iba odpor R , ostatné boli určené voľným odhadom. Pre stavový priestor bol začiatočný model vytvorený príkazom `idss`. Ako vstupné argumenty príkaz očakáva matice modelu, začiatočné podmienky stavových veličín a v prípade diskrétneho modelu, vzorkovaciu periódnu. V prípade spojitého modelu je posledný argument



Obr. 4.2: Výsledok filtrovania signálu odberu elektrického prúdu

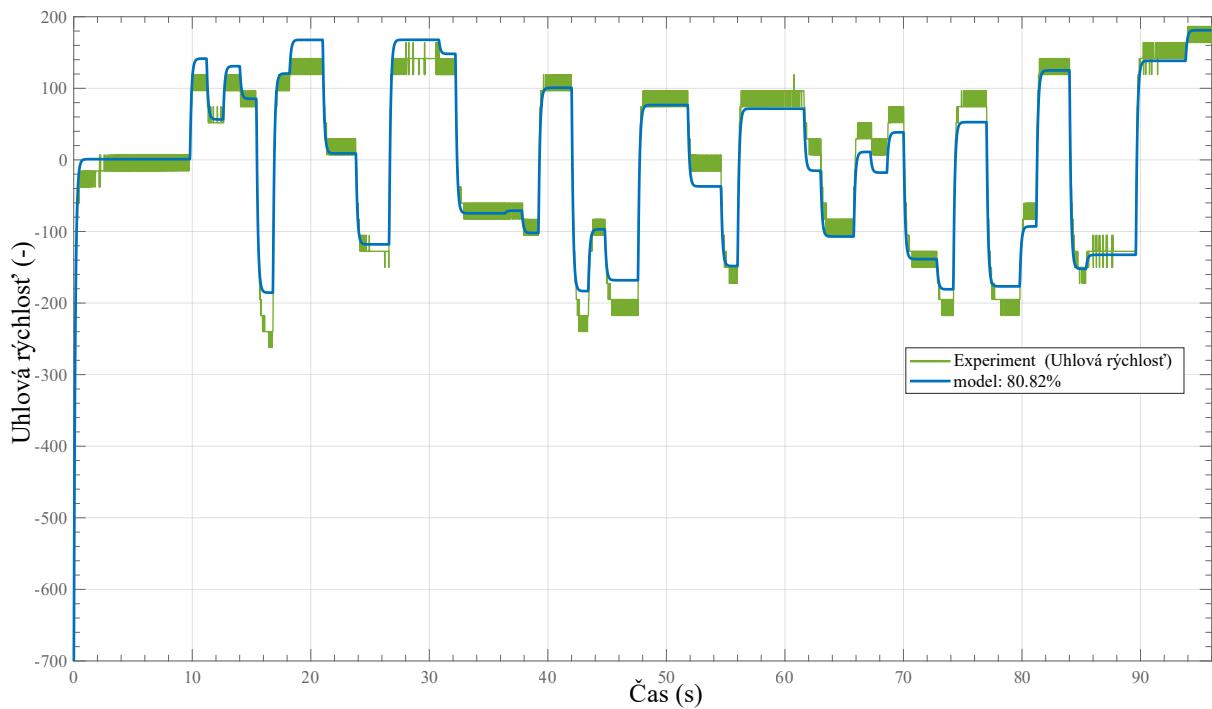
číslo 0. Po vytvorení modelu je dôležité určiť ktoré parametre bude algoritmus identifikovať a ktoré sú pevné konštanty. Toto je nastavené definovaním vlastnosti `Free` pre príslušnú maticu. Napríklad, pre maticu **C** sú všetky prvky konštantné a nechceme ich identifikovať. Toto je zabezpečené príkazom `sys.Structure.C.Free=0`. Pred spustením algoritmu identifikácie sú jeho parametre nastavené objektom triedy `ssestOptions`. Algoritmus identifikácie pre stavový priestor je spustený funkciou `ssest`, kde prvým argumentom je dátový objekt, druhým je začiatočný model a posledným je objekt nastavení. Začiatočný model prenosovej funkcie je vytvorený príkazom `idtf`. Prvý argument je vektor koeficientov polynómu čitateľa a druhý vektor koeficientov charakteristického polynómu. Algoritmus identifikácie je spustený funkciou `tfest`, vstupné argumenty sú rovnaké ako pri funkcií `ssest`. Algoritmus identifikácie bol zvolený automatický pre obidve funkcie. Zdrojové kódy identifikácie prenosovej funkcie a stavového priestoru, spolu s programom na zber dát sú dostupné v prílohe C. Identifikovaná prenosová funkcia má tvar:

$$G(s) = \frac{1.326 \cdot 10^5}{s^2 + 84.28s + 649.9}. \quad (4.12)$$

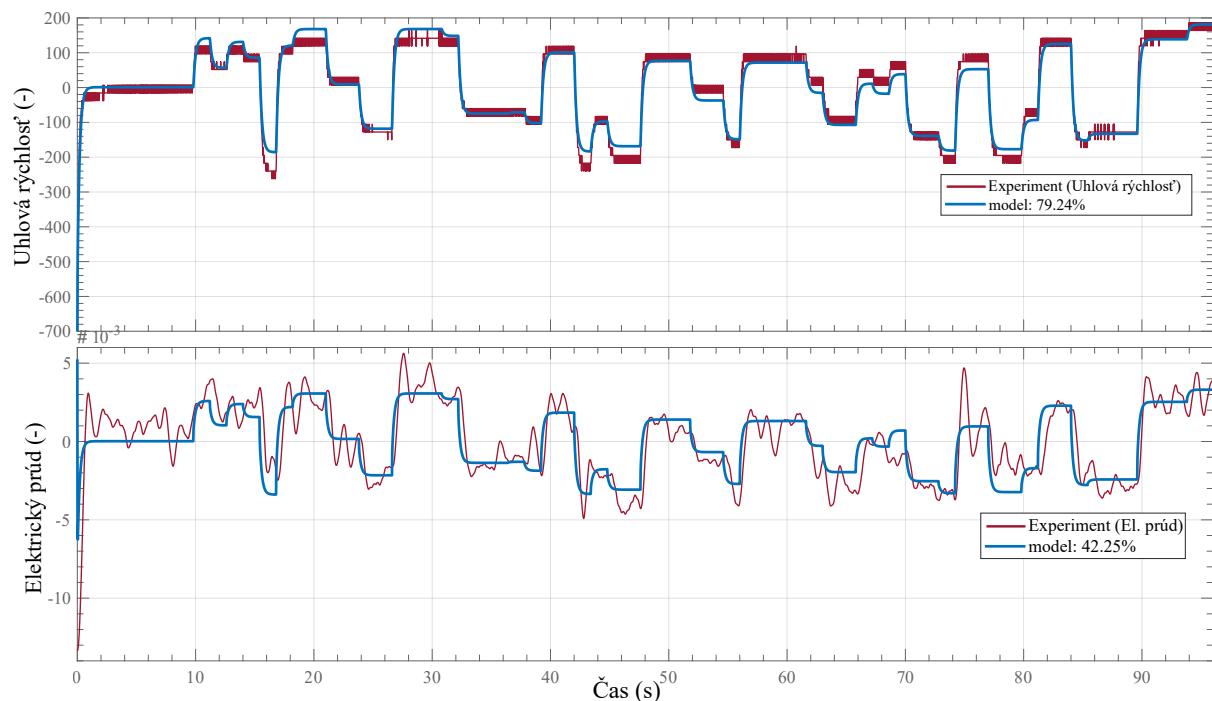
Matice stavovej rovnice identifikovaného stavového modelu nadobudli nasledovné hodnoty:

$$A = \begin{bmatrix} -21.19 & 1.162 \cdot 10^6 \\ 1.251 \cdot 10^{-3} & -97.26 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 0.1082 \end{bmatrix}. \quad (4.13)$$

Úspešnosť identifikácie je možné skontrolovať porovnaním identifikovaného modelu s dátami. Porovnané dáta s prenosovou funkciou je sú znázornené na Obr. 4.3 a so stavovým priestorom na Obr. 4.4.



Obr. 4.3: Porovnanie identifikovanej prenosovej funkcie s nameranými hodnotami



Obr. 4.4: Porovnanie identifikovaného stavového modelu s nameranými hodnotami

5 Nasledovná iterácia hardvéru modulu

Prvá verzia modulu MotoShield bola navrhnutá v práci [12], kde autor navrhoval dva didaktické moduly. Predpokladám, že nedostatky navrhnutého MotoShieldu nastali dôsledkom nedostatku času a nepozornosti. Tieto nedostatky môžeme skorigovať minimálnymi úpravami.

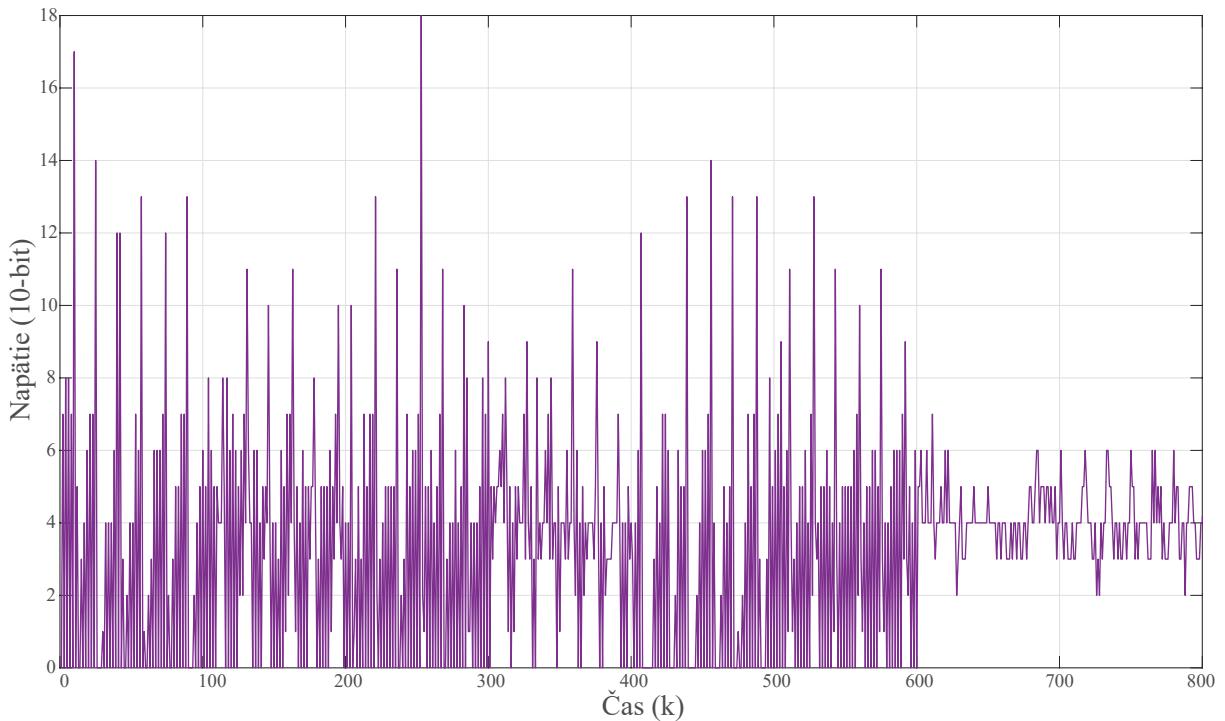
Prvý nedostatok sa vyskytol pri kresbe schémy zapojenia, keď autor navrhol spoj medzi kanálmi inkrementálneho snímača a digitálnymi pinmi D3 a D4. Pin D4 nie je vhodný na externé prerušenie, keď hovoríme o doskách Arduino UNO, Mega 2560, Leonardo a Zero[3]. Namiesto D4 je správne pripojiť kanál snímača na pin D2. Na meranie uhlovnej rýchlosťi pomocou MATLAB API je toto nevyhnutné, preto pri použití prvej verzie hardvéru je potrebné pin D4 fyzicky prepojiť káblom s pinom D2.

Ďalší nedostatok hardvéru je nekompatibilita modulu s doskami ktoré používajú 3.3 V CMOS logickú úroveň. Nekompatibilita spočíva v tom, že je signál z Hallovej sondy „binarizovaný“ Schmittovím preklápacím obvodom s hornou úrovňou 5 V. Horná hodnota výstupného signálu zo Schmittovho obvodu, z kanálu snímača je rovná hodnote napájacieho napäťia. Dosky s 3.3 V logickou úrovňou sa poškodia, ak na digitálny vstup priviedieme napätie s hodnotou 5 V. V opačnom prípade, ak priviedieme 3.3 V signál na digitálny vstup mikroradiča s 5 V TTL logickou úrovňou, mikroradič ho prečíta ako logickú jednotku. Presnejšie, mikroradič s 5 V logickou úrovňou prečíta ako logickú jednotku napätie hodnoty 3 až 5 V [5]. Preto je vhodné snímač napájať z 3.3 V pinu mikroradiča.

Posledná funkčná úprava je výmena periférie na nepriame meranie elektrického prúdu. V prvej verzii hardvéru sa prúd meria pomocou meracieho rezistora a dvoch operačných zosilňovačov. Prvý zosilňovač má za úlohu odčítať napätie pred a po meracom rezistore, kým druhý zosilňovač má za úlohu tento signál zosilniť. Signál je zosilnený o 2.96 (-), podľa rovnice (1.1). Toto je veľmi malé zosilnenie, dôkazom je priebeh signálu vyčítaného z 10-bitového analógovo-digitálneho prevodníka, Obr. 5.1. Tieto čísla predstavujú hodnotu napäťia, ktorá sa prevedie na volty násobkom $\frac{5}{1023}$ V. Správne zosilnený signál by mal nadobúdať maximálnu hodnotu (3-4.5) V.

Namiesto dvoch operačných zosilňovačov, v nasledujúcej verzii hardvéru navrhujem použiť čip INA169. Tento čip slúži práve na nepriame meranie prúdu. Dostupný je iba v SOT-23 prevedení s piatimi terminálmi. Prvky s prevedením SOT-23 sú na tlačenú obvodovú dosku pripojené technológiou SMT¹. Čip má päť terminálov, z ktorých GND a V+ sú na napájanie a terminály VIN+ a VIN- predstavujú vstupy operačného zosilňovača medzi ktoré sa umiestní merací rezistor R_s . Terminál OUT vysiela analógový signál, ktorého napätie je priamoúmerné hodnote prúdu. Odpór R_l je zapojený k výstupnému signálu paralelne na zem. Hodnota meracieho odporu R_s určuje pásmo prúdu a hodnota

¹abbrev. Surface Mount Technology, slov. povrchová montáž



Obr. 5.1: Vyčítaný signál odberu prúdu z 10-bitového analógovo-digitálneho prevodníka

odporu R_l určuje zosilnenie. Po zmeraní hodnoty výstupného signálu sa skutočná hodnota prúdu dostane pomocou vzťahu:

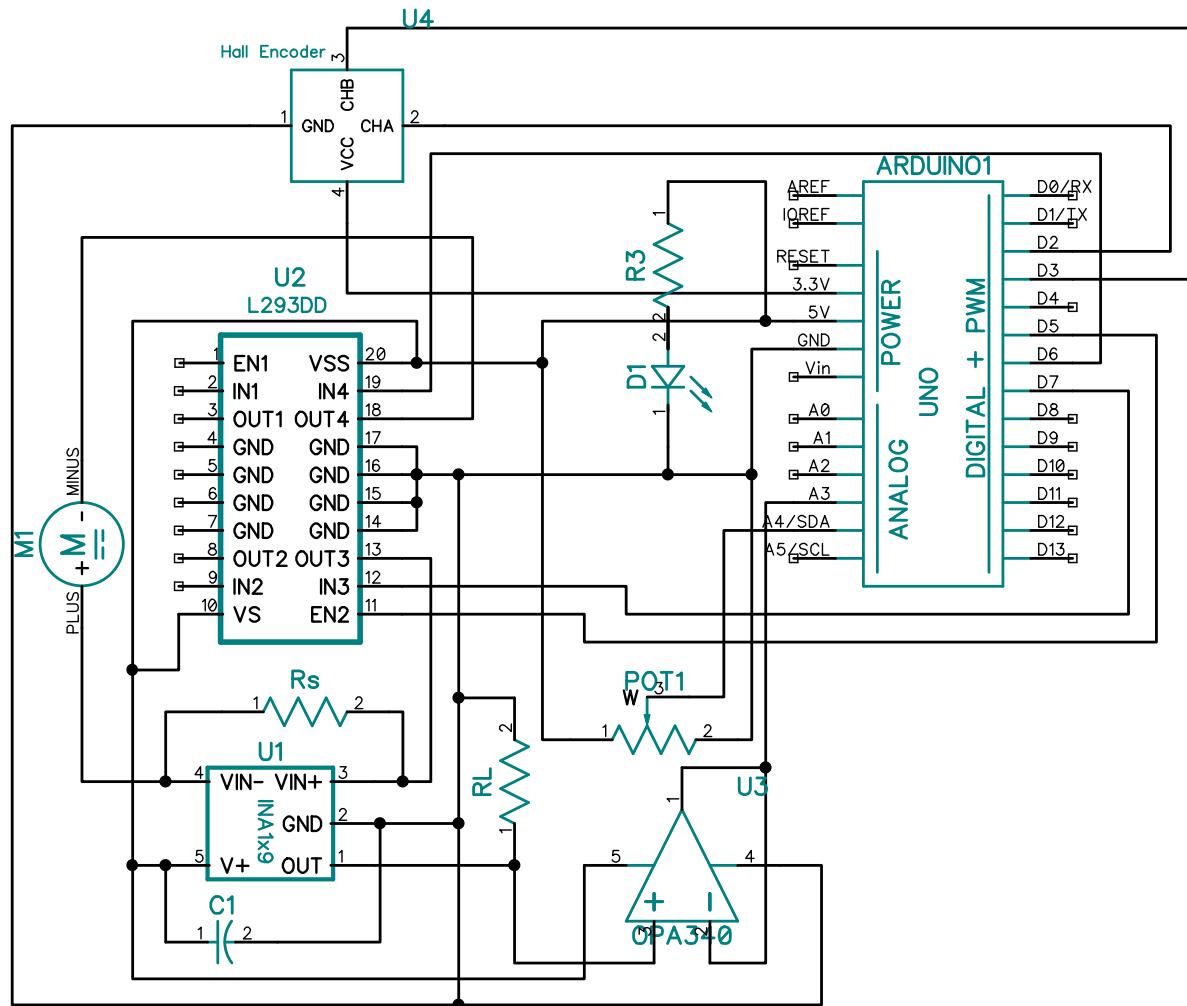
$$I_s = \frac{V_{OUT} \cdot 1k\Omega}{R_s \cdot R_l}. \quad (5.1)$$

Na dosiahnutie čím vyššieho stupňa presnosti je v technickom liste [21] odporúčané použiť tzv. „buffer“ zosilňovač ako je napr. OPA340. Zapojený medzi AD (analógovo-digitálnym) prevodníkom a výstupným terminálom čipu INA169 so spätnou väzbou, eliminuje vplyv impedancie AD prevodníka na zosilnenie. Impedancia AD prevodníka pôsobí paralelne na odpor R_l . Kompletná zostava periférie na meranie prúdu je schematicky znázornená na Obr. 5.2.

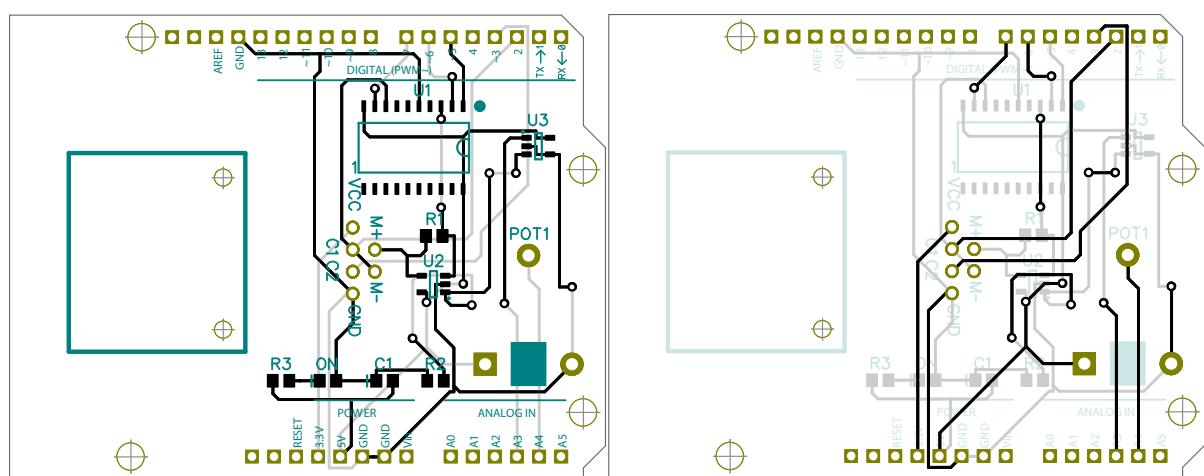
V prvej verzii hardvéru je navrhnutý H-mostík L293D v prevedení PDIP-16, pripojený k plošnému spoju technológiou THT². Montáž pomocou tejto technológie si vyžaduje dierku, cez ktorú sa pretiahne vývod súčiastky a zaleje sa spájkou. Z dôvodu osobnej preferencie do nového hardvéru navrhnenom rovnakú súčiastku, L293DD, v prevedení SO-20, ktoré patrí do skupiny súčiastok montovaných technológiou SMT. Hoci súčiastka v SO-20 prevedení má o štyri terminály viac ako v PDIP-16 prevedení, súčiastka nemá viacej funkcií, nadbytočné vývody sú prepojené so zemou.

Navrhnutý elektrický obvod, Obr. 5.2, bol testovaný na inom motore, bez inkrementálneho snímača, a ukázal sa ako funkčný. Pri testovaní bola použitá integrovaná prototypizačná doska s INA169 čipom, R_l a R_s odpormi s hodnotami 10 kΩ a 10 Ω. Prototypizačná doska zahŕňa aj kondenzátor s hodnotou 0.1 μF na vyhľadenie napájacieho napäťia pre INA169. V testovacom obvode bol zahrnutý aj H-mostík L293D.

²abbrev. Through-hole Technology



Obr. 5.2: Schéma elektrického obvodu ďalšej iterácie modulu



Obr. 5.3: Tlačená obvodová doska - PCB

6 Záver

Príkladmi v Kap. 3 som potvrdil funkčnosť programátorského prostredia pre MotoShield, pre vývojové prostredia Arduino IDE, MATLAB a Simulink. Tu boli napísané aj príklady pre spätnoväzbové riadenie pomocou PID regulátora, ktorého parametre som sice dostał heuristicky, ale výsledkom je rýchly a dostatočne presný regulačný obvod. Identifikáciou systému bola získaná prenosová funkcia, podľa ktorej bol vytvorený blok v prostredí Simulink. Pomocou tohto bloku je možné simulovať reguláciu uhlovej rýchlosťi MotoShieldu aj bez fyzického modulu.

Pri identifikácii boli získané modely vo forme prenosovej funkcie a stavovej reprezentácie. Porovnaním odozvy modelu a nameraných dát pre uhlovú rýchlosť bola získaná približne osemdesiat percentná zhoda, kým pre prúd približne štyridsať percentná. Pri prúde bola odchýlka spôsobená hlavne zašumiením signálu, ktorý som sa neúspešne viacerými spôsobmi snažil vyfiltrovať. Parametre identifikovaného modelu majú, žiaľ, nereálne hodnoty v porovnaní s fyzikálnymi veličinami systému. Pre lepšie výsledky identifikácie je odporúčané skúsiť používať novú hardvérovú zostavu na meranie prúdu. Tešíme sa na zoznamenie s metódami riadenia, ktoré využívajú matematický model systému.

V ďalšej iterácii programátorských rozhranií je očakávané rozšírenie funkčností MotoShieldu. Primárne by som sa sústredil na prípady zaťaženia alebo pridania zotrvačníka na hriadeľ. Pre tieto prípady nie sú, v práci zmienené, kalibračné metódy vhodné. Bolo by potrebné navrhnúť nové kalibračné funkcie, ktoré by mali možnosť uvedenia pridaného momentu zotrvačnosti, čo by spôsobilo zmenu hodnôt časových oneskorení vo vnútri funkcie. Taktiež by bolo vitané napísať knižnicu pre Arduino IDE, ktorá bude zahŕňať metódy na digitálne spracovanie signálov, prevažne IIR filtre.

Z dôvodu neprimeraných pracovných podmienok (prebiehajúcej epidemiologickej krízy), ďalšia iterácia hardvéru nebola sfinalizovaná. Pred výrobou modulu je dôležité elektrické zapojenie otestovať a prípadne niektoré prvky vymeniť. Napísané programátorské rozhrania je potom potrebné upraviť a prispôsobiť novému hardvéru.

Literatúra

- [1] Arduino. ARDUINO UNO REV3. Online Store. [cit. 27.4.2020], <https://store.arduino.cc/arduino-uno-rev3>.
- [2] Arduino. Writing a library for arduino. Online. [cit. 27.4.2020], <https://www.arduino.cc/en/Hacking/LibraryTutorial>.
- [3] Arduino Reference. External interrupts. Online. [cit. 27.4.2020], <https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/>.
- [4] Arduino Reference. Variable scope qualifiers - Volatile. Online. [cit. 11.6.2020], <https://www.arduino.cc/reference/en/language/variables/variable-scope-qualifiers/volatile/>.
- [5] Arduino Reference. Defining Pin Levels: HIGH and LOW. Online, 7.2003. [cit. 11.6.2020], <https://www.arduino.cc/reference/en/language/variables/constants/constants/>.
- [6] C. Belavý. *Základy automatizácie a merania*. Slovenská technická univerzita v Bratislave, Bratislava, Vazovova 5, 2012.
- [7] CodinGame. Inline variables - 7 Features of C++17 that will simplify your code. Online. [cit. 11.6.2020], <https://www.codingame.com/playgrounds/2205/7-features-of-c17-that-will-simplify-your-code/inline-variables>.
- [8] DFRobot. Micro Metal Geared motor w/Encoder. Online Store. [cit. 27.4.2020], <https://www.dfrobot.com/product-1437.html>.
- [9] G. Takács, M. Gulán. Automationshield repository. Online GitHub. [cit. 4.6.2020], <https://github.com/gergelytakacs/AutomationShield>.
- [10] Gergely Takács. AutomationShield API Style Guide. Online. [cit. 27.4.2020], <https://github.com/gergelytakacs/AutomationShield/issues/42>.
- [11] Isaac Physics. Kirchhoff's Laws. Online. [cit. 11.6.2020], https://isaacphysics.org/concepts/cp_kirchhoffs_laws.
- [12] T. Konkoly. Experimentálne moduly pre výučbu automatizácie. Slovenská technická univerzita v Bratislave, 5 2018. Bakalárská práca, [cit. 27.4.2020].

- [13] MathWorks. Set Arduino pin mode - MATLAB configurePin. Online. [cit. 27.4.2020], <https://www.mathworks.com/help/supportpkg/arduinoio/ref/configurepin.html#bus7qlz-1-mode>.
- [14] MathWorks. Simulink documentation. Online. [cit. 4.6.2020], <https://www.mathworks.com/help/simulink/>.
- [15] MathWorks MATLAB Hardware Team. MATLAB Support Package for Arduino Hardware. Online. [cit. 27.4.2020], <https://www.mathworks.com/matlabcentral/fileexchange/47522-matlab-support-package-for-arduino-hardware>.
- [16] P. Prinz and U. Kirch-Prinz. *A Complete Guide to Programming in C++*. Jones and Bartlett Publishers, Sudbury, Massachusetts, 2002.
- [17] Quanser. QNET 2.0 DC Motor Board. Online. [cit. 11.6.2020], <https://www.quanser.com/products/qnet-2-0-dc-motor-board/>.
- [18] Rick Hill. System Dynamics and Control - Introduction to State-Space Modeling. Online Video, 29.1.2015. [cit. 4.6.2020], <https://www.youtube.com/watch?v=X3TOZLJCWiY>.
- [19] STMicroelectronics. L293d push-pull four channel driver with diodes. Online Datasheet, 7.2003. [cit. 11.6.2020], <https://www.st.com/resource/en/datasheet/1293d.pdf>.
- [20] G. Takács, M. Gulan, J. Bavlna, R. Köplinger, M. Kováč, E. Mikuláš, S. Zarghoon, and R. Salíni. HeatShield: a low-cost didactic device for control education simulating 3D printer heater blocks. In *Proceedings of the 2019 IEEE Global Engineering Education Conference (EDUCON)*, pages 374–383, Dubai, United Arab Emirates, April 2019.
- [21] Texas Instruments. INA1x9 high-side measurement current shunt monitor. Online Datasheet, 2.2017. [cit. 11.6.2020], <https://www.ti.com/lit/ds/symlink/ina169.pdf?ts=1591356334230>.
- [22] W3Schools. C++ classes and objects. Online, 23.4.2017. [cit. 4.6.2020], https://www.w3schools.com/cpp/cpp_classes.asp.

A Programátorské rozhrania - API

A.1 Arduino IDE API - Hlavičkový súbor

```
1 #ifndef MOTOSHIELD_H_
2 #define MOTOSHIELD_H_
3 #include "Arduino.h"
4 #define MOTO_YPIN1      3
5 #define MOTO_YPIN2      4
6 #define MOTO_UPIN       5
7 #define MOTO_DIR_PIN1   6
8 #define MOTO_DIR_PIN2   7
9 #define MOTO_RPIN       A4
10 #define MOTO_YV1         A0
11 #define MOTO_YV2         A1
12 #define MOTO_YAMP1      A2
13 #define MOTO_YAMP2      A3
14 #define AMP_GAIN        2.96
15 #define SHUNT           10.0
16 class MotoShieldClass{
17     public:
18         void setDirection(bool direction = true);
19         void begin(float _Ts = 50.0);
20         void calibration();
21         void actuatorWrite(float percentValue);
22         void actuatorWriteVolt(float voltageValue);
23         float referenceRead();
24         float sensorReadRPM();
25         float sensorReadRPMperc();
26         float sensorReadVoltage();
27         float sensorReadVoltageAmp1();
28         float sensorReadVoltageAmp2();
29         float sensorReadCurrent();
30         static inline volatile uint8_t count;
31         static inline volatile uint8_t counted;
32         static inline volatile bool stepEnable;
33         uint32_t minRPM, minDuty, maxRPM;
34         float minVolt;
35     private:
36         static void _InterruptServiceRoutine();
37         static void _InterruptSample();
38         float _K;
39     };
40 extern MotoShieldClass MotoShield;
41 #endif
```

A.2 Arduino IDE API - Zdrojový súbor

```
1 #include "MotoShield.h"
2 #include "AutomationShield.h"
3 #include "Sampling.h"
4 void MotoShieldClass::_InterruptServiceRoutine(){
5     count++;
6 }
7 void MotoShieldClass::_InterruptSample(){
8     counted = count;
9     count = 0;
10    stepEnable = true;
11 }
12 void MotoShieldClass::setDirection(bool direction = true) {
13     if (direction) {
14         digitalWrite(MOTO_DIR_PIN1, 0);
15         digitalWrite(MOTO_DIR_PIN2, 1);
16     }
17     else {
18         digitalWrite(MOTO_DIR_PIN1, 1);
19         digitalWrite(MOTO_DIR_PIN2, 0);
20     }
21 }
22 void MotoShieldClass::begin(float _Ts = 50.0){
23     pinMode(MOTO_UPIN, OUTPUT);
24     pinMode(MOTO_DIR_PIN1,OUTPUT);
25     pinMode(MOTO_DIR_PIN2,OUTPUT);
26     pinMode(MOTO_YPIN1, INPUT);
27     pinMode(MOTO_YPIN2, INPUT);
28     pinMode(MOTO_RPIN, INPUT);
29     pinMode(MOTO_YV1, INPUT);
30     pinMode(MOTO_YV2, INPUT);
31     pinMode(MOTO_YAMP1, INPUT);
32     pinMode(MOTO_YAMP2, INPUT);
33     setDirection(true);
34     Sampling.period(_Ts*1000.0);
35     _K=60000.0/_Ts;
36     Sampling.interrupt(_InterruptSample);
37     attachInterrupt(digitalPinToInterrupt(MOTO_YPIN1), _InterruptServiceRoutine, CHANGE);
38 }
39 float MotoShieldClass::referenceRead() {
40     return AutomationShield.mapFloat((float)analogRead(MOTO_RPIN), 0.0, 1023.0, 0.0, 100.0);
41 }
42 void MotoShieldClass::actuatorWrite(float percentValue) {
43     if(percentValue < minDuty && percentValue != 0) percentValue = minDuty;
44     analogWrite(MOTO_UPIN, AutomationShield.percToPwm(percentValue));
45 }
46 void MotoShieldClass::actuatorWriteVolt(float voltageValue){
47     if(voltageValue < minVolt != 0) voltageValue = minVolt;
48     analogWrite(MOTO_UPIN,sq(voltageValue)*255.0/sq(AREF));
49 }
50 void MotoShieldClass::calibration(){
51     actuatorWrite(100);
52     delay(1000);
53     maxRPM = counted;
54     actuatorWrite(0);
55     delay(500);
56     int i = 15;
57     do{
58         actuatorWrite(i);
59         delay(300);
60         if(counted >= 4){
61             delay(1000);
62             minDuty = i;
63             minRPM = counted;
64             minVolt = AREF * sqrt(minDuty/100.0);
65             actuatorWrite(0);
66             break;
67         }
68     }
```

```

68         i++;
69     }while(1);
70 }
71 float MotoShieldClass::sensorReadRPMperc(){
72     return AutomationShield.constrainFloat(AutomationShield.mapFloat(counted, (float)minRPM, (float)maxRPM,
73     → 0.0, 100.0),0.0,100.0);
74 }
75 float MotoShieldClass::sensorReadRPM(){
76     return (float)counted/14.0*_K;
77 }
78 float MotoShieldClass::sensorReadVoltage(){
79     return fabs((analogRead(MOTO_YV2)-analogRead(MOTO_YV1))*5.0/1023.0);
80 }
81 float MotoShieldClass::sensorReadVoltageAmp1(){
82     return analogRead(MOTO_YAMP1)*5.0/1023.0;
83 }
84 float MotoShieldClass::sensorReadVoltageAmp2(){
85     return analogRead(MOTO_YAMP2)*5.0/1023.0/AMP_GAIN;
86 }
87 float MotoShieldClass::sensorReadCurrent(){
88     return MotoShield.sensorReadVoltageAmp2()/SHUNT*1000.0;
89 }
MotoShieldClass MotoShield;

```

A.3 MATLAB API

```

1  classdef MotoShield < handle
2      properties(Access = public)
3          arduino;
4          encoder;
5          minDuty = 25;
6          minRPM;
7          maxRPM;
8      end
9      properties(Constant)
10         MOTO_YPIN1 = 'D2';
11         MOTO_YPIN2 = 'D3';
12         MOTO_UPIN = 'D5';
13         MOTO_DIR_PIN1 = 'D6';
14         MOTO_DIR_PIN2 = 'D7';
15         MOTO_YV1 = 'A0';
16         MOTO_YV2 = 'A1';
17         MOTO_YAMP1 = 'A2';
18         MOTO_YAMP2 = 'A3';
19         MOTO_RPIN = 'A4';
20         SHUNT = 10.0;
21         AMP_GAIN = 2.96;
22         PPR = 7;
23     end
24     methods(Access = public)
25         function begin(MotoShieldObject, Port, Board)
26             MotoShieldObject.arduino = arduino(Port,Board,'Libraries','rotaryEncoder');
27             configurePin(MotoShieldObject.arduino,MotoShieldObject.MOTO_YPIN1,'Interrupt');
28             configurePin(MotoShieldObject.arduino,MotoShieldObject.MOTO_YPIN2,'Interrupt');
29             configurePin(MotoShieldObject.arduino,MotoShieldObject.MOTO_RPIN,'AnalogInput');
30             configurePin(MotoShieldObject.arduino,MotoShieldObject.MOTO_YV1,'AnalogInput');
31             configurePin(MotoShieldObject.arduino,MotoShieldObject.MOTO_YV2,'AnalogInput');
32             configurePin(MotoShieldObject.arduino,MotoShieldObject.MOTO_YAMP1,'AnalogInput');
33             configurePin(MotoShieldObject.arduino,MotoShieldObject.MOTO_YAMP2,'AnalogInput');
34             configurePin(MotoShieldObject.arduino,MotoShieldObject.MOTO_UPIN,'PWM');
35             configurePin(MotoShieldObject.arduino,MotoShieldObject.MOTO_DIR_PIN1,'DigitalOutput');
36             configurePin(MotoShieldObject.arduino,MotoShieldObject.MOTO_DIR_PIN2,'DigitalOutput');
37             MotoShieldObject.encoder = rotaryEncoder(MotoShieldObject.arduino,MotoShieldObject.MOTO_YPIN1]
→             ,MotoShieldObject.MOTO_YPIN2,MotoShieldObject.PPR);
38             setDirection(MotoShieldObject);
39             disp('MotoShield initialized.')
40         end
41         function setDirection(MotoShieldObject, direction)
42             if nargin < 2
43                 direction = 1;
44             end
45             if direction
46                 writeDigitalPin(MotoShieldObject.arduino,MotoShieldObject.MOTO_DIR_PIN1,0);
47                 writeDigitalPin(MotoShieldObject.arduino,MotoShieldObject.MOTO_DIR_PIN2,1);
48             else
49                 writeDigitalPin(MotoShieldObject.arduino,MotoShieldObject.MOTO_DIR_PIN1,1);
50                 writeDigitalPin(MotoShieldObject.arduino,MotoShieldObject.MOTO_DIR_PIN2,0);
51             end
52         end
53         function actuatorWrite(MotoShieldObject, percentValue)
54             if (percentValue < MotoShieldObject.minDuty) && (percentValue ~= 0)
55                 percentValue = MotoShieldObject.minDuty;
56             end
57             writePWMDutyCycle(MotoShieldObject.arduino, MotoShieldObject.MOTO_UPIN, (percentValue / 100));
58         end
59         function ref = referenceRead(MotoShieldObject)
60             ref = readVoltage(MotoShieldObject.arduino,MotoShieldObject.MOTO_RPIN) * 100 / 5;
61         end
62         function voltageDrop = sensorReadVoltage(MotoShieldObject)
63             voltageDrop = readVoltage(MotoShieldObject.arduino,MotoShieldObject.MOTO_YV1)-readVoltage(Mot]
→             oShieldObject.arduino,MotoShieldObject.MOTO_YV2);
64         end

```

```

65     function voltageDropAmp1 = sensorReadVoltageAmp1(MotoShieldObject)
66         voltageDropAmp1 = readVoltage(MotoShieldObject.arduino,MotoShieldObject.MOTO_YAMP1)*100/5;
67     end
68     function voltageDropAmp2 = sensorReadVoltageAmp2(MotoShieldObject)
69         voltageDropAmp2 =
70             readVoltage(MotoShieldObject.arduino,MotoShieldObject.MOTO_YAMP2)/MotoShieldObject.AMP_GAIN * 100 / 5;
71     end
72     function current = sensorReadCurrent(MotoShieldObject)
73         current = voltageDropAmp2 / MotoShieldObject.SHUNT * 1000.0;
74     end
75     function calibration(MotoShieldObject)
76         writePWMDutyCycle(MotoShieldObject.arduino, MotoShieldObject.MOTO_UPIN, 1);
77         pause(1);
78         MotoShieldObject.maxRPM = readSpeed(MotoShieldObject.encoder);
79         writePWMDutyCycle(MotoShieldObject.arduino, MotoShieldObject.MOTO_UPIN, 0);
80         pause(0.5);
81         i = 0.15;
82         resetCount(MotoShieldObject.encoder);
83         while(1)
84             writePWMDutyCycle(MotoShieldObject.arduino, MotoShieldObject.MOTO_UPIN, i);
85             pause(0.3);
86             if abs(readCount(MotoShieldObject.encoder)) > 8
87                 pause(1);
88                 MotoShieldObject.minDuty = i*100;
89                 MotoShieldObject.minRPM = readSpeed(MotoShieldObject.encoder);
90                 actuatorWrite(MotoShieldObject,0);
91                 break;
92             end
93             i=i+0.01;
94         end
95         function rpm = sensorReadRPM(MotoShieldObject)
96             rpm = readSpeed(MotoShieldObject.encoder);
97         end
98         function rpmPerc = sensorReadRPMPerc(MotoShieldObject)
99             rpmPerc = constrain(map(readSpeed(MotoShieldObject.encoder),MotoShieldObject.minRPM,MotoShieldObject.
100                maxRPM,0,100),0,100);
101            end
102        end

```

A.4 Zdrojový kód MATLAB funkcie pre blok MotoSHield - Simulink API

```
1 function [Actuator, Y] = fcn(Ref,Calibration,ExternalLow,ExternalUp,Time,Sensor)
2 persistent i t timeMem Upper Lower WasCalibrated;
3 if isempty(Upper)
4     Lower = 0;
5     Upper = 0;
6     i = 40;
7     t=0;
8     WasCalibrated = 0;
9 end
10 Actuator = 0;
11 if (Calibration == 1) && (WasCalibrated == 0)
12     if Time < 2.1
13         Actuator = 255;
14         if Time >= 2
15             Upper = Sensor(2);
16         end
17     end
18     if (Time >= 3)
19         Actuator = i;
20         if Time >= 3+t
21             t = t + 0.1;
22             i = i + 1;
23         end
24         if Sensor(2) > 400
25             i = i - 1;
26             if isempty(timeMem)
27                 timeMem = Time;
28             end
29             if Time >= timeMem + 1.5
30                 Lower = Sensor(2);
31                 WasCalibrated = 1;
32             end
33         end
34     end
35 end
36 if (Calibration ~= 1) && (Upper == 0)
37 Upper = ExternalUp;
38 Lower = ExternalLow;
39 WasCalibrated = 1;
40 end
41 if(WasCalibrated == 1)
42     Actuator = Ref*2.55;
43 end
44 Y = constrain(map(Sensor(2),Lower,Upper,0,100),0,100);
```

B PID riadenie

B.1 Arduino IDE

```
1 #include <MotoShield.h>
2 #include <PIDAbs.h>
3 #define TS 40.0
4 #define AUTO 1
5 #define KP 0.000001
6 #define TI 0.0003
7 #define TD 0.001
8 float r = 0.0;
9 float R[]={40.0,70.0,50.0,85.0,35.0,60.0};
10 float y = 0.0;
11 float u = 0.0;
12 unsigned int k = 0;
13 int T = 80;
14 int i = 0;
15 void setup() {
16     Serial.begin(2000000);
17     MotoShield.begin(TS);
18     MotoShield.calibration();
19     PIDAbs.setKp(KP);
20     PIDAbs.setTi(TI);
21     PIDAbs.setTd(TD);
22     PIDAbs.setTs(TS);
23 }
24 void loop() {
25     if (MotoShield.stepEnable) {
26         step();
27         MotoShield.stepEnable=false;
28     }
29     void step(){
30     #if !AUTO
31         r = MotoShield.referenceRead();
32     #else AUTO
33         if(i >= sizeof(R)/sizeof(float)){
34             MotoShield.actuatorWrite(0.0);
35             while(true);
36         }
37         if (k % (T*i) == 0){
38             r = R[i];
39             i++;
40         }
41         k++;
42     #endif
43     y = MotoShield.sensorReadRPMperc();
44     u = PIDAbs.compute(r-y,0,100,0,100);
45     MotoShield.actuatorWrite(u);
46     Serial.print(r);
47     Serial.print(" ");
48     Serial.println(y);
49 }
```

B.2 MATLAB

```
1 clear
2 close all
3 clc
4 Manual = 0;
5 MotoShield = MotoShield;
6 MotoShield.begin('COM4','UNO');
7 MotoShield.calibration()
8 PID = PID;
9 Ts = 0.04;
10 k = 1;
11 stepEnable = 0;
12 R = [80, 70, 50, 85, 35, 60];
13 T = 100;
14 i = 0;
15 Kp = 0.007;
16 Ti = 0.002;
17 Td = 0.1;
18 PID.setParameters(Kp, Ti, Td, Ts);
19 tic
20 while (1)
21     if (toc >= Ts * k)
22         stepEnable = 1;
23     end
24     if (stepEnable)
25         if(Manual == 0)
26             if (mod(k, T*i) == 1)
27                 i = i + 1;
28                 if (i > length(R))
29                     MotoShield.actuatorWrite(0.0);
30                     break
31                 end
32                 r = R(i);
33             end
34         end
35         if(Manual == 1)
36             r = MotoShield.referenceRead();
37         end
38         y = MotoShield.sensorReadRPMperc();
39         u = PID.compute(r-y, 0, 100, 0, 100);
40         MotoShield.actuatorWrite(u);
41         response(k, :) = [r, y, u];
42         k = k + 1;
43         stepEnable = 0;
44     end
45 end
46 save response response
47 disp('The example finished its trajectory. Results have been saved to "response.mat" file.')
48 plotPIDResponse('response.mat',1)
```

C Identifikácia

C.1 Stavový model - MATLAB

```
1 clc; clear; close all;
2 load aprbsResult.mat
3 Ts = 0.02;
4 data = iddata([y ifltr],u,Ts,'Name','Experiment');
5 data = detrend(data);
6 data.InputName = 'Input Voltage';
7 data.InputUnit = 'V';
8 data.OutputName{1} = 'Angular Velocity';
9 data.OutputUnit{1} = 'rad/s';
10 data.OutputName{2} = 'Current';
11 data.OutputUnit{2} = 'A';
12 data.Tstart = 0;
13 data.TimeUnit = 's';
14 J = 0.01;
15 b = 0.1;
16 Ke = 1;
17 Kt = 1;
18 R = 10;
19 L = 3;
20 dtheta0 = data.y(1,1);
21 i0 = data.y(1,2);
22 A = [-b/J      Kt/J;
23           -Kt/L     -R/L];
24 B = [ 0;
25       1/L];
26 C = [1 0;
27         0 1];
28 D = [0;
29       0];
30 K = zeros(2,2);
31 x0 = [dtheta0; i0];
32 disp('Initial guess:')
33 sys = idss(A,B,C,D,K,x0,0)
34 sys.Structure.A.Free = [1 1;
35           1 1];
36 sys.Structure.B.Free = [0
37           1];
38 sys.Structure.C.Free = false;
39 sys.DisturbanceModel = 'estimate';
40 sys.InitialState = 'estimate';
41 Options = ssestOptions;
42 Options.Display = 'on';
43 Options.Focus = 'simulation';
44 Options.SearchOptions.MaxIterations = 50;
45 Options.InitialState = 'estimate';
46 disp('Estimated model:')
47 model = ssest(data,sys,Options);
48 compare(data,model)
49 grid on
50 save MotoShield_GreyboxModel_LinearSS model
```

C.2 Prenosová funckia - MATLAB

```
1 clc; clear; close all;
2 load aprbsResult.mat
3 Ts = 0.02;
4 data = iddata(y,u,Ts,'Name','Experiment');
5 data = detrend(data);
6 data.InputName = 'Input Voltage';
7 data.InputUnit = 'V';
8 data.OutputName = 'Angular Velocity';
9 data.OutputUnit = 'rad/s';
10 data.Tstart = 0;
11 data.TimeUnit = 's';
12 J = 0.01;
13 b = 0.01;
14 Ke = 1;
15 Kt = 1;
16 R = 10.3;
17 L = 0.1;
18 b0 = Kt;
19 a2 = J*L;
20 a1 = (J*R+b*L);
21 a0 = (b*R+Kt^2);
22 disp('Initial model:')
23 sys = idtf([b0],[a2 a1 a0])
24 sys.Structure.Denominator.Maximum = [1000 1000 inf];
25 sys.Structure.Denominator.Minimum = [0 0 0];
26 sys.Structure.Numerator.Maximum= inf;
27 sys.Structure.Numerator.Minimum= 0;
28 Options = tfestOptions;
29 Options.Display = 'on';
30 Options.InitialCondition = 'estimate';
31 Options.SearchOptions.MaxIterations = 100;
32 disp('Estimated model:')
33 model = tfest(data,sys,Options)
34 zmodel = c2d(model,Ts);
35 compare(data,model);
36 grid on
37 save MotoShield_GreyboxModel_TF model zmodel
```

C.3 Zber dát - Arduino IDE

```
1 #include<MotorShield.h>
2 #define TS 20.0
3 #define PRBS 0
4 #if PRBS
5 #include "prbsU.h"
6 float prbs;
7 #else
8 #include "aprbusU.h"
9 float aprbs;
10 #endif
11 int i;
12 float u;
13 void setup() {
14     Serial.begin(500000);
15     MotorShield.begin(TS);
16 }
17
18 void loop() {
19     if (MotorShield.stepEnable) {
20         step();
21         MotorShield.stepEnable = false;
22     }
23 }
24 void step() {
25 #if PRBS
26     if (i > prbsU_length) {
27         MotorShield.actuatorWrite(0);
28         while (1);
29     }
30     else {
31         u = pgm_read_word(&prbsU[i]);
32     }
33 #else
34     if (i > aprbsU_length) {
35         MotorShield.actuatorWrite(0);
36         while (1);
37     }
38     else {
39         u = pgm_read_float_near(&aprbusU[i]);
40     }
41 #endif
42     MotorShield.actuatorWriteVolt(u);
43     Serial.print(u,5);
44     Serial.print(", ");
45     Serial.print(MotorShield.counted);
46     Serial.print(", ");
47     Serial.println(MotorShield.sensorReadCurrent(),5);
48     i++;
49 }
```